九十年度
國防科技發展方案學術合作研究計劃

題目

HLA 在兵棋演訓 Federate 的類別程式庫之研究與設計

90-2623-7-032-003

軍方研發需求單位：中科院系發中心兵棋模擬組
計畫執行單位：淡江大學資訊工程學系
主持人：黃俊堯 副教授
研究助理：杜怡璋
汪洋

計畫時程：自 90 年 1 月 1 日起至 90 年 12 月 31 日止

目　錄

第一章、研究目的

在現代戰爭中，每一個參與戰爭的個體(包括人及武器系統)能夠獨立作戰又能相互協調的能力，是贏得此戰爭的重要關鍵。為達到此目標，定期舉行大型軍事演習是有其重要性與必要性，然而每次舉辦此類型軍事演習所需消耗的人力、財力、及物力是相當龐大，隨著軍費預算的縮減、衛星科技的進步、及環保意識的高漲，使得此類大型軍事演習的進行日益困難。

因此，歐美等各先進國家一直在尋求另一種演習方式，冀求在經濟支援與演習效果之間得到一個平衡點，而模擬器與電腦兵棋則是提供不同層級的解決方案。然而模擬器與電腦兵棋的應用層級與發展技術背景有很大的差異性。由於彼此的差異性，在以往的發展歷史中，模擬器與電腦兵棋的網路聯模演訓技術分成兩個集團各自發展，無法將兩者發展的技術作有效的整合。直到 1995 年，美國國防部提出一個新的模式模擬架構，稱為高階模擬架構(High Level Architecture－HLA)，方才正式將模擬器與電腦兵棋整合於相同的網路環境進行聯合模擬演訓，而今日的高階模擬架構已成為新一代的分散式網路互動模擬標準。圖 1-1 是美國國防部的網路聯合模擬演訓技術之研發歷史。

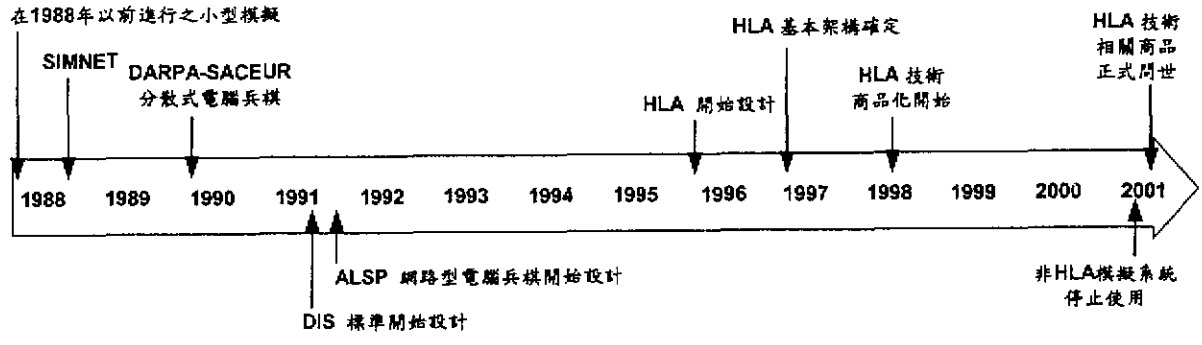


圖 1-1、美國網路模擬技術發展史

由圖 1-1 得知，在美國網路模擬演訓技術發展歷程中，重要的里程碑包括 1988 年所制定的 SIMNET(SIMulation NETwork)規格來串聯模擬器系統以初步驗證網路聯合模擬的可行性與成本效益，在 SIMNET 成功的達成目的後，美國國防部進而在 1991 年延伸出 DIS(Distributive Interactive Simulation)規格來完整的規範串聯模擬器系統及真實武器進行聯合模擬演訓所需之標準。但是由於

DIS 技術僅規範即時模擬系統的規格，對於需要多種時間管理技術的分析性電腦兵棋系統卻無法處理，於是美國 DARPA (Defense Advanced Research Projects Agency)委託 MITRE 公司研議以 DIS 與 SIMNET 技術發展適合集合層次的戰術性模擬(Aggregate Level Constructive Simulation)系統，且在 1991 年 1 月選用了一套稱為 GRWS/M 陸軍兵棋模擬系統來開始測試一套專屬電腦兵棋系統的網路互動技術 - ALSP (Aggregate Level Simulation Protocol)。

雖然 DIS 技術成功地串連模擬器來進行互動，而 ALSP 技術使得不同電腦兵棋系統之間可以進行聯網互動，有效的提供不同作戰層級的解決方案，但是 DIS 與 ALSP 技術皆無法使得模擬器與電腦兵棋系統之間串連在一起進行聯合模擬演訓，於是美國國防部的 Defense Modeling and Simulation Office (簡稱 DMSO)在 1995 年十月依據所訂定的模式模擬(Modeling and Simulation -- M&S)主導性計畫(DoD 5000.59-P)提出了高階模擬架構(High Level Architecture - HLA)的概念。此主導性計畫是由美國國防部的 DMSO 來主導整個標準的制定與修改，其目的是要致力於建構一個公開通用的技術架構組織以促進其所有型式的模組及模擬系統(Models and Simulations)與 C4I 系統做溝通互動，亦即是在促進 M&S 元件重複使用性。而這個公開通用的技術架構組織則包含了高階模擬架構(HLA)，且在美國國防部的 M&S 發展領域中，HLA 的發展具有最高的優先權。

HLA 之主要目的為整合各種型態的模擬系統（包括真人操作真實系統、模擬器系統和兵棋軟體）以及提升模擬系統各元件之重複使用性與相容性。在 HLA 的環境下，所有模擬物件，不管是兵棋軟體，還是真實武器系統，統稱為 Federates。在內容上，HLA 包括了以下三個部份：

- 高階模擬架構之規則(The HLA Rules)
- 高階模擬架構之介面定義(The HLA Interface Specification)
- 高階模擬架構之物件模型格式(The HLA Object Model Template)

其中，HLA Rules 共有十條，是用來描述在建立聯模演訓時，各個演習物件(Federate)程式設計的基本規範、Federate 如何與 OMT 結合、以及 Federate 如

何透過 Interface Specification 來進行 Federates 之間的互動關係。而其中的五個規則是用來定義 Federation 的設立條件，另外五個規則則是用來定義 Federate 與 Federate 之間的關係。HLA 的 Interface Specification 是定義在 HLA 架構下，規範 Federates 之間進行互動所應有的服務類別與介面呼叫函式，其中可分成六大管理模組：

- 演習管理程式模組(Federation Management)

  此介面規格定義演習的產生與關閉，以及一個 federate 如何動態地加入與離開進行中的聯模演訓。

- 宣告管理程式模組(Declaration Management)

  此介面規格定義 federate 如何對其它 federate 宣告它所有產生的物件屬性資料，以及向其它 federate 宣告所想要接收之資料的方法。在 HLA 環境下，federate 必須先向其它 federate 告知其所要產生或接收之物件屬性資料，方可在聯合模擬演習當中傳送與接收資料。

- 物件管理程式模組(Object Management)

  此介面規格定義了在聯合模擬演習進行當中，federate 之間互相傳遞物件屬性與互動關係的方法，以及動態產生與刪除屬性資料的功能。

- 使用權管理程式模組(Ownership Management)

  此介面規格使得 federate 之間可以在聯合模擬演習進行當中相互轉移物件屬性的擁有權與使用權。

- 時間管理程式模組(Time Management)

  此介面規格負責處理 federate 之間的時間同步協調，使得聯合模擬演訓進行當中所發生的所有事件能保持應有的因果關係。

- 空間分割管理模組(Data Distribution Management)

此介面規格使得 federate 可以定義屬性資料的影響範圍，使得聯合模擬演訓可以更精確地模擬真實世界狀況，例如：武器的有效半徑或雷達的偵搜範圍等。

HLA 更進一步定義了實作上述介面規格的軟體，名稱為執行基礎(Run - Time Infrastructure，簡稱 RTI)。根據 HLA 的規範，RTI 是個分散式作業系統，每一個 HLA 環境下的 federate 都必須透過 RTI 來與其它 federate 溝通。值得一提的是，HLA 規格內一再強調，HLA 只定義了介面規格，並沒有規定 RTI 的製作方式，也就是說，任何人都可以用不同方式來製作 RTI，只要其 RTI 的呼叫函式符合 HLA 介面規格的定義即可，因此，美國已有數所大學投入 RTI 的設計工作，例如：美國麻省理工學院(MIT)、史丹佛大學、布朗大學等。

物件模型格式(OMT)是高階模擬架構(HLA)的第三部份元件，而在 HLA 中何以要有 OMT 呢？其基本原因可歸納如下：

- 提供一個簡單易懂之機制以描述在模擬演習中，所有成員之公用資料的交換情形。
- 提供一套標準化之機制以描述模擬演習成員之內定基本功能。
- 可促進發展 HLA 物件資料庫之共通工具集之應用程式的設計。

基本上，物件模型格式是架構在物件導向之上的，所以 OMT 會延襲物件導向的基本觀念及特性，但並非全部，其中最重要的一項是繼承(Inheritance)這個特性。依使用的時機與需求的不同，可以將 OMT 的應用分成下列兩項：Federation Object Models (FOM)與 Simulation Object Models (SOM)。

在一個聯模演訓(Federation)中只會有一個 FOM 存在，而 FOM 的最主要目的是在於提供一個共通且標準化之格式來描述在一個聯模演訓(Federation)中各模擬器(Federate)之間資料的交換情形。而這些需要在聯模演訓(Federation)中交換的資料則包括：所有與此演訓有直接關係之物件及互動類別，所有描述這些物件特徵之屬性，及所有描述這些互動所需之參數。

但是在一個模擬演習(Federation)中，會存在一個以上的SOM，而SOM的最主要目的在於描述聯模演訓中之個別模擬器(Federate)可提供給整個演習的功能。而SOM也和FOM一樣是用物件類別、互動類別、物件屬性及互動參數來定義。由於FOM和SOM的格式是相同的，所以，SOM可以成為FOM的一部分，亦即是說，在發展FOM的過程中，可以先發展出所有的SOM，再將之合併成一個FOM。模擬系統在HLA環境中與FOM、SOM及Runtime Infrastructure之間的關係可以用圖1-2來表示。

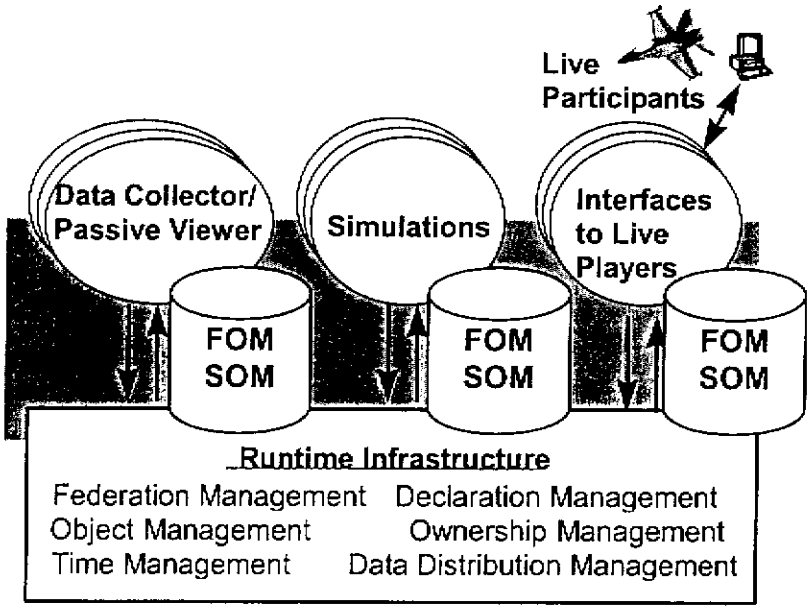


圖 1-2、HLA 環境之基本架構

HLA 在美國已經發展一段時日，而且現已趨於完整成熟的階段。由於 HLA 的目的是在建構一個聯模演訓(Federation)環境，所以聯模演訓的發展步驟便成為 HLA 中重要的一項工作。然而在 HLA 的文件中都只定義了基本規格，發展者若欲從這些基本定義中去完全發展一個聯模演訓(Federation)是一件相當困難的事，所以各個模訓發展者都極力在研究如何推演出一個有效地方法來設計 HLA 聯模演訓。

基於這個需求，DMSO 在 1997 年 11 月提出一套標準的聯模演訓發展與執行程序稱為 Federation Development and Execution Process (FEDEP)，用以協助 HLA 的系統發展人員設計與發展聯模演訓，並順利產出演訓資料庫及相關模擬系統元件。不過 FEDEP 只是一種導引說明文件，其用意在於建議一個發展

Federation 的逐步過程可供聯模演訓發展者參考使用，並不是一套完整的發展方法，雖然 FEDEP 只是扮演引導者的角色，但是仍有其相當的存在價值，以下便就 FEDEP 的內容做一概述。

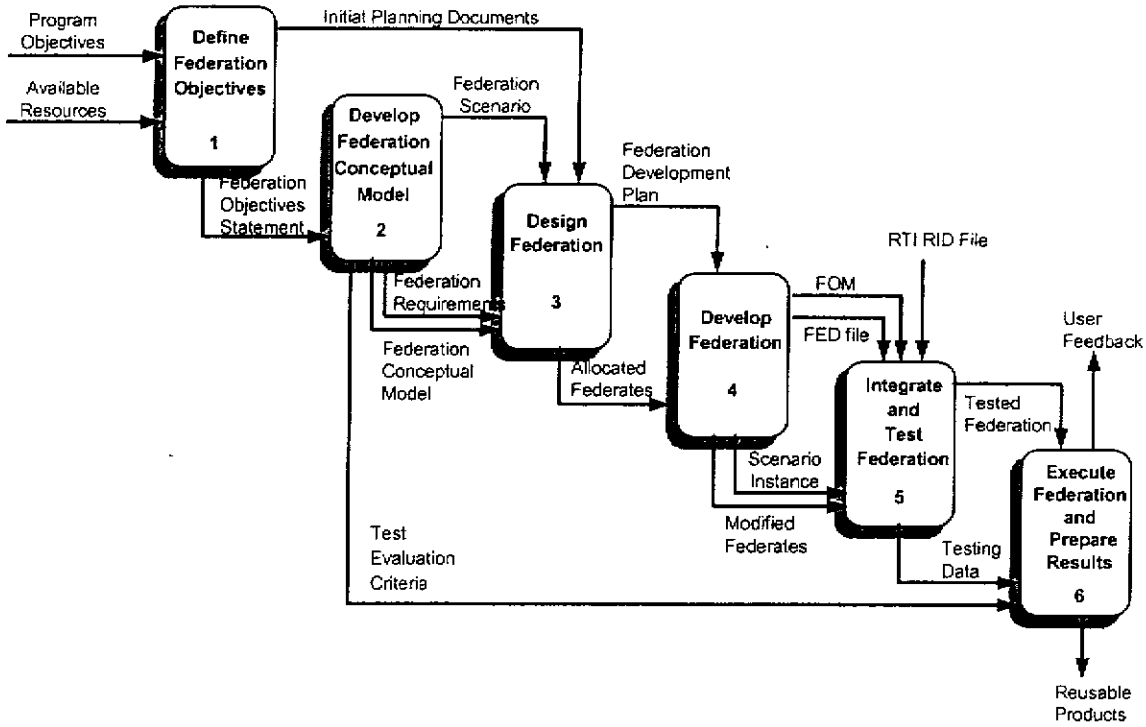


圖 1-3、FEDEP 的六大發展步驟

在 FEDEP 中，基本上將模擬演訓(Federation)的發展步驟分成如圖 1-3 所示的六大步驟，這六大步驟分述如下：

[步驟一]定義聯模演訓目標(Define Federation Objective)

模擬演訓的贊助者及發展者必須要共同研討出此模擬演訓的目的及需求，且記錄要完成這些目的所需完成的事項。

[步驟二]發展聯模演訓之概念模型(Develop Federation Conceptual Model)

設計足以代表真實世界中所要模擬之事物的物件模型，並且發展文件來描述該物件的表示方式以及互動關係。

[步驟三]設計聯模演訓(Design Federation)

決定所有參與這個聯模演訓的個別模擬器及模擬系統，並且制定出聯模演訓的發展計劃。

[步驟四]發展聯模演訓(Develop Federation)

根據步驟三所設計的聯模演訓計劃，發展新的模擬單元或是修改現有的模擬單元，並且制定出一個定義此模擬演訓的 FOM。

[步驟五]聯模擬演訓的整合與測試(Integrate and Test Federation)

所有聯模演訓的需求都發展完成，並加以測試，以便得知所發展出的聯模演訓是否達到所定義的需求。

[步驟六]聯模演訓執行及結果分析(Execute Federation and Analysis Results)

發展完成的模擬演訓經過執行後，發展者必須根據執行的輸出結果做分析研究，並將結果回饋給模擬演訓的贊助者。

在 FEDEP 規範中，除了定義了六大發展步驟外，還針對各大步驟建議其每個步驟的細節子步驟，並細分其各個子步驟的工作項目，而細分之工作項目關係對應如下表所示。

定義聯模演訓目標	發展聯模演訓之概念模型	設計聯模演訓	發展聯模演訓	聯模演訓的整合與測試	聯模演訓執行及結果分析
(1).確定需求 (2).確立目標	(1).發展想定 (2).實施概念分析 (3).發展模擬想定需求	(1).選定模擬系統 (2).指派功能 (3).準備計劃	(1).發展 FOM (2).建立模擬演訓協定 (3).實施模擬單元修改	(1).規劃執行步驟 (2).整合聯模演訓 (3).測試聯模演訓	(1).計劃執行 (2).處理輸出 (3).備便結果

總之，在美國國防部對於 HLA 技術不遺餘力的推廣情況下，不但此技術透過國際認證成為世界標準，目前已有包含日本、中國大陸、及歐美諸國採用 HLA 技術為其兵棋模擬系統的標準，而近年也正式有 HLA 技術相關的產品開

始銷售。反觀國內，對於 HLA 相關技術仍在討論當中，更不用談相關系統的研發。

而在 HLA 規格中有兩個核心技術是必須及早進行相關研究與發展，首先是模擬訓練所須之模訓資料庫(FOM/SOM)，其次是以 HLA 為基礎之電腦兵棋 federate 系統。其中以電腦兵棋 federate 系統最為複雜且耗人力，因此在美國已有相關學術機構與公司嘗試開發其類別程式庫，以期縮短以 HLA 為基礎之電腦兵棋 federate 系統的發展時程。因此，本計劃的目的就是在於提出開發以 HLA 為基礎之電腦兵棋 federate 軟體所須之系統架構，以及設計以 HLA 為基礎之電腦兵棋 federate 軟體所須之類別程式庫。藉由提出此系統架構下之程式庫可以作為未來開發電腦兵棋 federate 軟體的設計規範，使得日後開發以 HLA 為基礎之電腦兵棋系統得以迅速且正確的發展。未來藉由此類別程式庫所設計之電腦兵棋軟體具有模組化、可重覆使用、及可交互運作等三大特性，此三大特性為目前軟體系統開發之潮流，此類別程式庫使得未來開發以 HLA 為基礎之電腦兵棋 federate 軟體的製作成本將大為降低。

第二章、RTI 1.3 之 federation 建構標準

2.1 RTI 執行架構概要

RTI 軟體可以在單獨的工作站或是透過網路環境執行，DMSO 的 RTI 軟體包括：RtiExec、FedExec 及 LibRTI 1.3 三個部份，其中 RtiExec 的執行則負責了一個 federation execution 的開始執行或是被終止執行。另外，每個執行中的 federate 則以個別或整體的 FedExec 為其對外機制，如圖 2-1 所示。RtiExec 是一個整體性的處理程序，每一個以 RTI 為基礎的模擬軟體必須經由這個程序來初始化 RTI 的元件，以建立 Federation 中的溝通機制。FedExec 則管理了一個 Federation，它允許 federates 的加入、離開以及 federate 個體之間的資料交換。

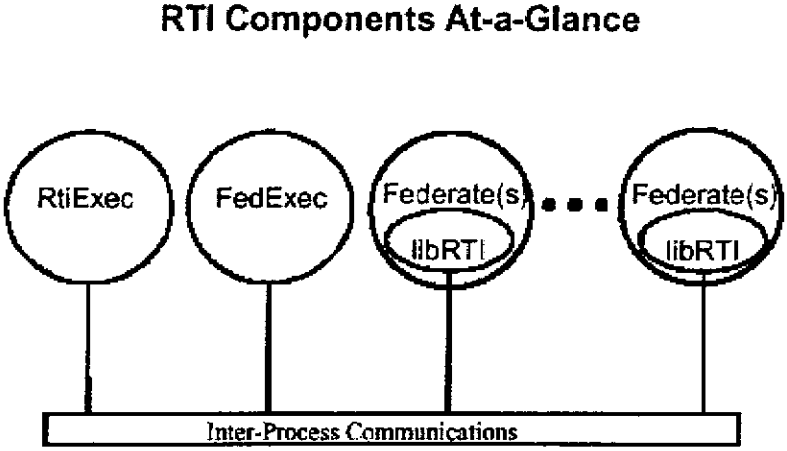


圖2-1

"libRTI" 是一個 C++ library，它提供了所有 HLA 介面函式規格所支援的 RTI 服務，將這些 Service 以介面函式的型態提供給 federation 的設計者使用，如圖 2-2 所示。透過 libRTI，RTIambassador class 將一些 RTI 所支援的 service 集中在一起，並由 federate 運用這些 method call 向 RTI 提出要求來執行 federation 的功能。相似的，FederateAmbassador 則是將 RTI 的 abstract call back function 集中並提供使用。

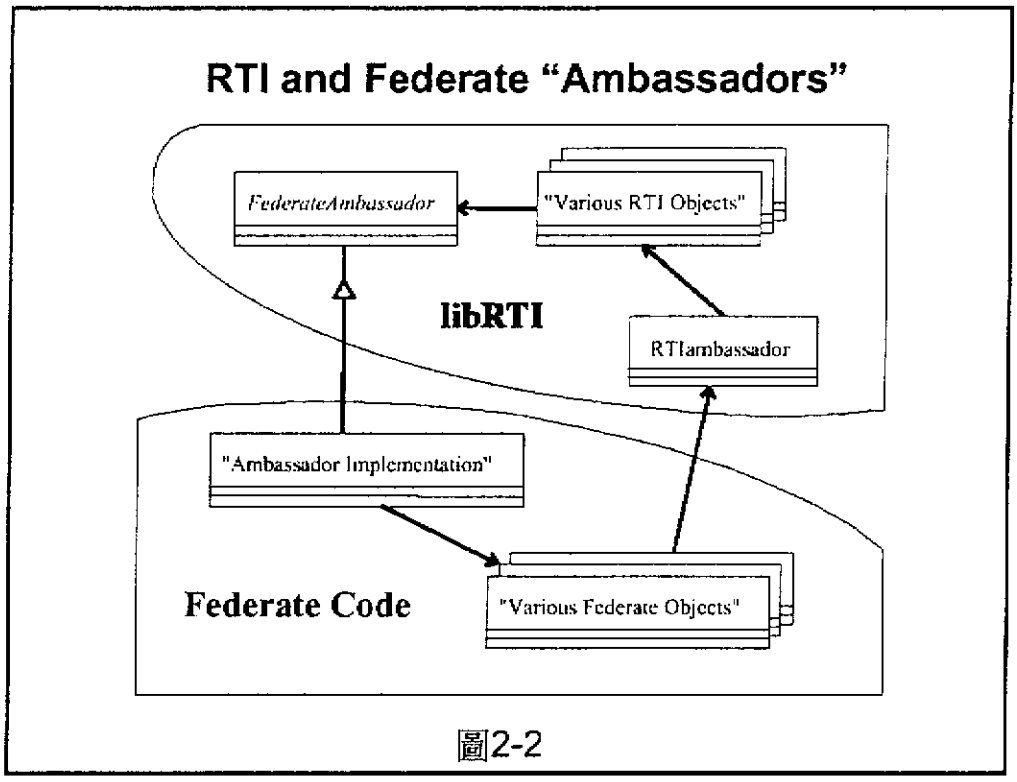


圖2-2

一個 Federation 的生命週期，從起始到結束有著一定的執行程序，以及一些必要的步驟，如圖 2-3 所示。而這些步驟分別位於 HLA 的六大管理模組，圖 2-4 為這六大管理模組在 Federation 執行階段中被呼叫使用的順序圖。在 Federation 起始的階段(start-up)，先使用 Federation management 提供的介面函式啟動 Federation 以及 federate 的加入，並做一些起始的設定，如：Synchronization point 的註冊；接下來，由 Declaration Management 所提供的介面函式宣告所有 federate 中的物件型態(屬於 Publish 還是 Subscribe)；完成這個動作後，Object Management 之介面函式開始創造物件實例；並以 Data distribution management 所提供的介面函式為物件實例加入場景分割的機制；在起始階段的最後一步，運用 Time management 所提供的介面函式宣告每個 federate 時間推進的方式。在執行階段(execution)中，六個模組的介面函式會分別在不同的情況下被執行，以 Federation management 來說，當 federate 需要儲存目前狀態時會被使用，或是 federate 在意外離開後要再次加入，也會被運用；如果要更改物件的宣告形態，則動態的呼叫 Declaration management 的介面函式；而執行階段中的訊息傳遞或 interaction 傳遞則透過 Object management 的介面函式來執行；當然，模擬時間的同步推進則由 Time management 的介面函

式負責；執行階段中，物件也會因為一些模擬的狀況改變而需要轉移被 federate 的擁有權，透過 Ownership management 的介面函式可以達到這個目的。最後，在模擬的情境都結束或已經到達規定完成時間，則進入結束狀態(end)，因為這個狀態下所有的動作都已經停止，因此 Ownership Management、Time management、Data distribution management 都不再被使用，而 Object Management 及 Declaration management 的介面函式處理物件移除及物件宣告的解除；最終再由 Federation management 的介面函式移除 federate 並結束 Federation。

在 RTI 1.3 Programmer Guide 中，以其執行時之功能詳細列出介面函式間的關聯與互動，並希望以此為依據，提供 federation 設計者或程式寫作者一個參考。在下一節中，將依照 Programmer Guide 列出這些介面函式的使用方式。

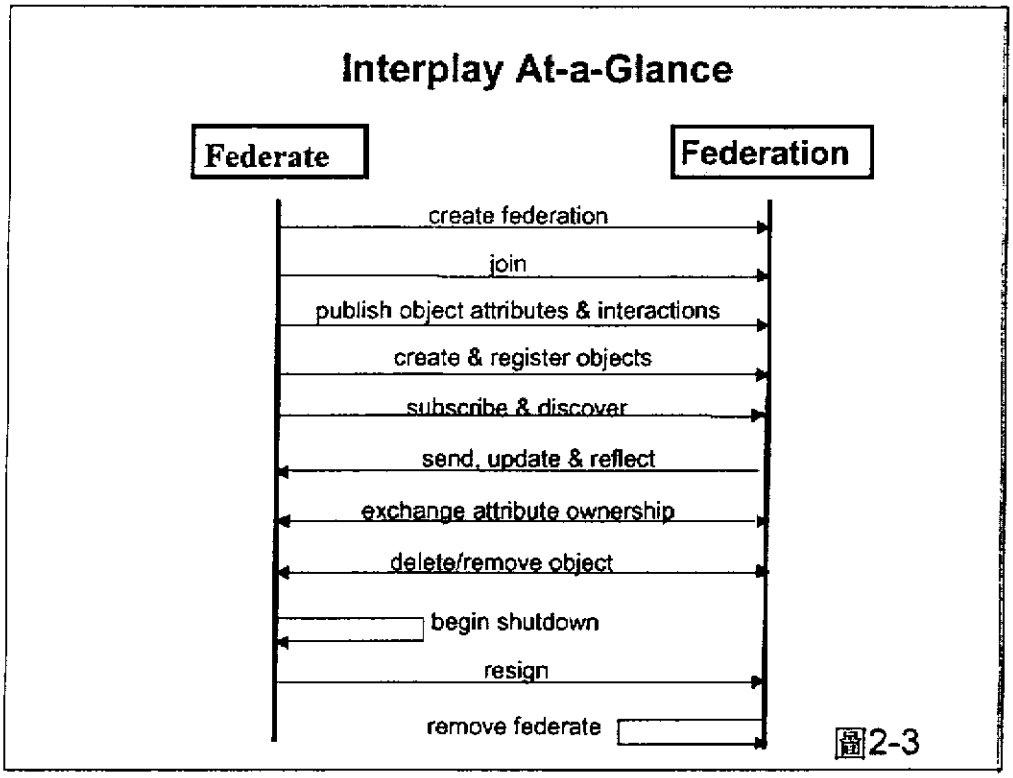


圖2-3

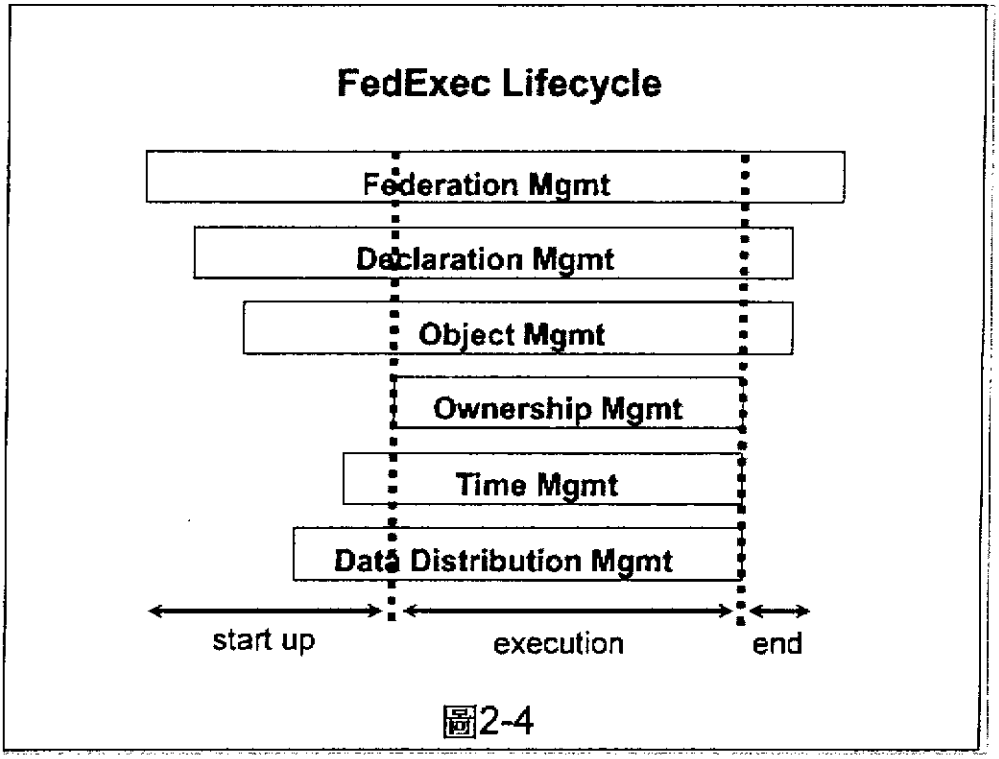

FedExec Lifecycle
Federation Mgmt
Declaration Mgmt
Object Mgmt
Ownership Mgmt
Time Mgmt
Data Distribution Mgmt
start up
execution
end

圖2-4

2.2 使用 RTI 1.3 介面函式 建立 Federation 之標準方式

在 RTI 1.3 之 Progrmmer's Guide 中，詳細的列出如何使用 RTI 所提供的介面函式以及每個介面函式的標準使用時機及方式。以下將針對六大管理模組中的介面函式來說明。

A、Federation Management

在 Federation Management 當中，所有介面函式可以分為幾個不同的功能群組，分別是：Life Cycle、Synchronization、Save、Restore，而為了執行這幾個主要功能，分別需要使用一些由 RTIambassador 對 RTI 提出執行要求的介面函式以及一些由 FedrateAmbassador 衍生之 instance 傳回的 call back。

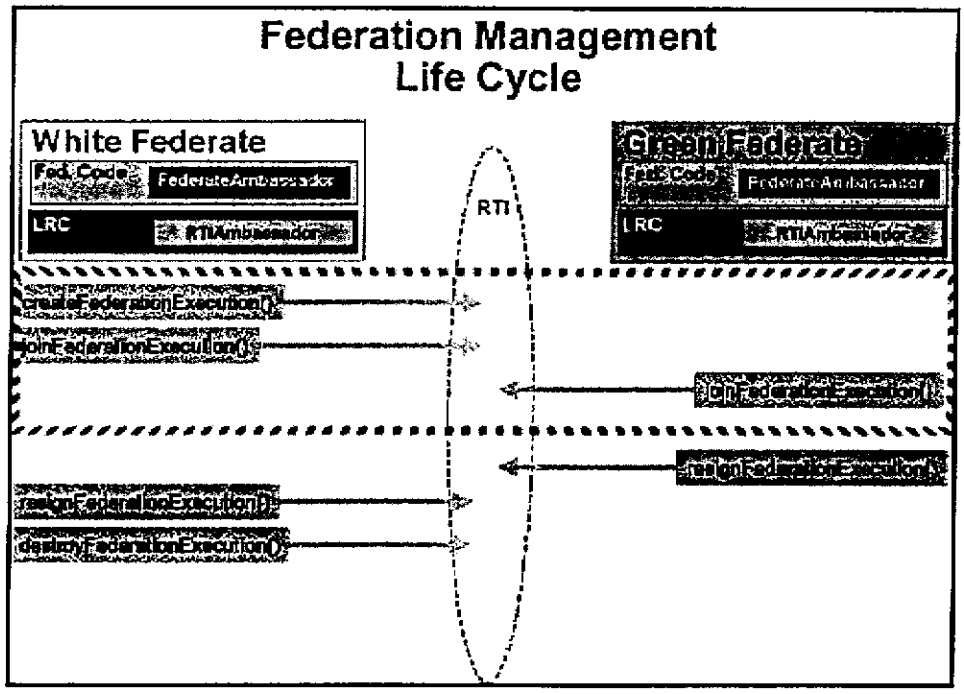


圖 2-5

圖 2-5 為以 Federation 之 Life Cycle 為主的幾個介面函式之執行方式，在創造一個 Federation 時，使用 creatFederationExecution()這個介面函式，接下來每一個 federate 個體要加入這個 Federation 時，以 joinFederationExecution()告知 RTI 要加入的要求。而當 federate 要退出 Federation 時，以 resignFederationExecution()來向 RTI 提出要求。最後，在一個 Federation 要結束

其模擬時，以 destroyFederationExecution()來告知 RTI。以這四個介面函式來說，federation 設計者必須要在起始及結束時使用。

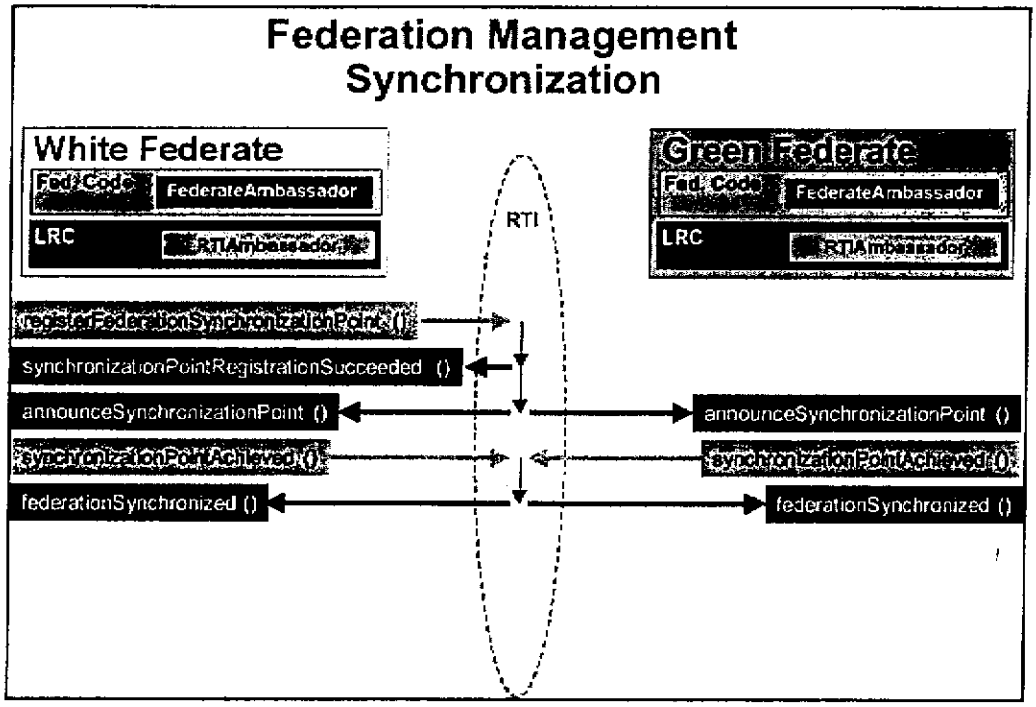


圖 2-6

圖 2-6 為 Federation 在起始後接著要執行的程序，運用 registerFederationSynchronization() 針對不同執行階段的幾個 Synchronization Point 做出註冊，當註冊成功後，RTI 會回傳一個 synchronizationPointRegistrationSucceeded()告知 federate 已經註冊成功。如果 federate 要求註冊的這個註冊點已經存在，則以 announceSynchronizationPoint() 通知 federate 已經被列入 List 當中，隨時可以宣告到達 Synchronization Point。所以當 federate 到達一個階段執行完畢時會先停止，以 synchronizationPointAchieved()通知 RTI 已經到達 Synchronization Point。而在同一個 Federation 中的所有 federates 都到達同一個 Synchronization Point 時，就由 RTI 以 federationSynchronized() callback 告訴每個 federates 可以向下一個階段推進。

圖 2-7 為 Federation 狀態在需要儲存時所需要用到的介面函式。當設計 federation 的人需要在某些環境狀態改變時預先做出 federation 整體狀態的儲存，則必須以 requestFederationSave() 來告知 RTI，此時 RTI 將以

initialFederateSave()來 call back 告知 federate 已停止所有的訊息傳遞，並等待 federate 執行狀態儲存。接下來 federate 以 federateSaveBegun()告知 RTI 開始執行儲存，當儲存完畢後，以 federateSaveComplete()通知 RTI 已經完成儲存。RTI 會以 federateSaved() call back 並通知開始恢復執行，或可以進行下一個動作。

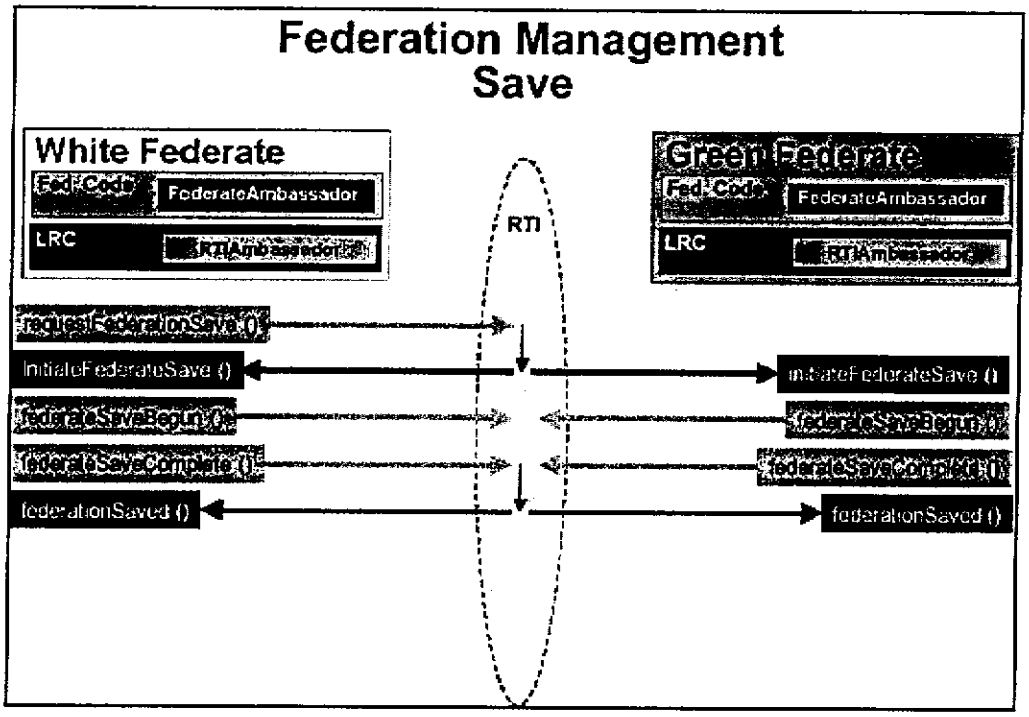


圖 2-7

圖 2-8 為 Federate 如果要 restore 的必須步驟，一個 federate 要提出 restore 的請求時以 requestFederationRestore() 告知 RTI，接著，RTI 會以 requestFederationRestoreSucceeded() call back 給提出要求 federate 表示已經收到要求，並以 federationRestoreBegun() call back 告知所有的 federate 開始 restore，另外，RTI 再以 initialFederateRestore() 通知所有的 Federate 開始設定 restore 狀態。當 federate restore 完成後，以 federateRestoreComplete()通知 RTI，當 RTI 獲知所有 federate 都已經完成 restore，則以 federationRestored()告知所有 federates 可以繼續執行。

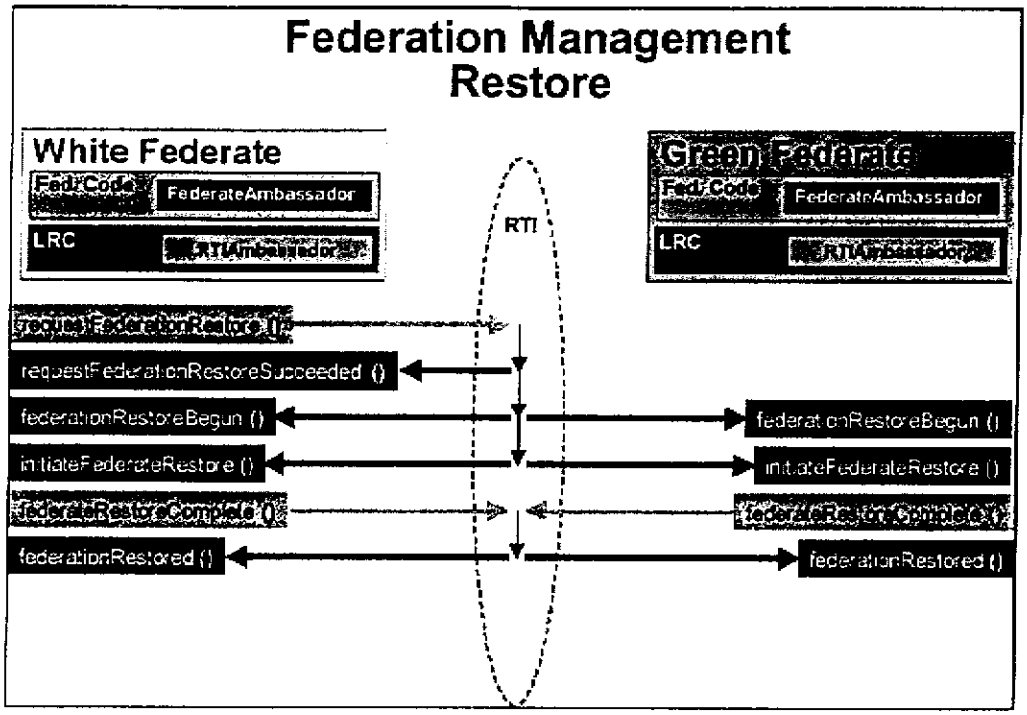


圖 2-8

B、Time Management

在 Time Management 的介面函式中，可分為：Policy、Time Step Advancement、Event-based Advancement、Optimistic Advancement、Queries 等不同功能的幾個群組。

圖 2-9 為針對設定 federates 的時間運作角色所需要用到的介面函式 s。如果 federation 設計者需要將一個 federate 設定為 regulation 模式(也就是主動時間的推進)，則使用 enableTimeRegulation()告知 RTI 做設定宣告，而 RTI 會以 timeRegulationEnabled() call back 告知宣告成功。如果 federate 在任意時間，必須要離開或更改時間控制屬性，則以 disableTimeRegulation()告知 RTI 放棄之前的宣告。同理，如果 federation 設計者需要將一個 federate 設定為 constrained 模式(也就是被動的推進時間)，則使用 enableTimeConstrained()告知 RTI 做設定宣告，而 RTI 會以 timeConstrainedEnabled() call back 告知宣告成功。如果 federate 在任意時間，必須要離開或更改時間控制屬性，則以 disableTimeConstrained() 告知 RTI 放棄之前的宣告。

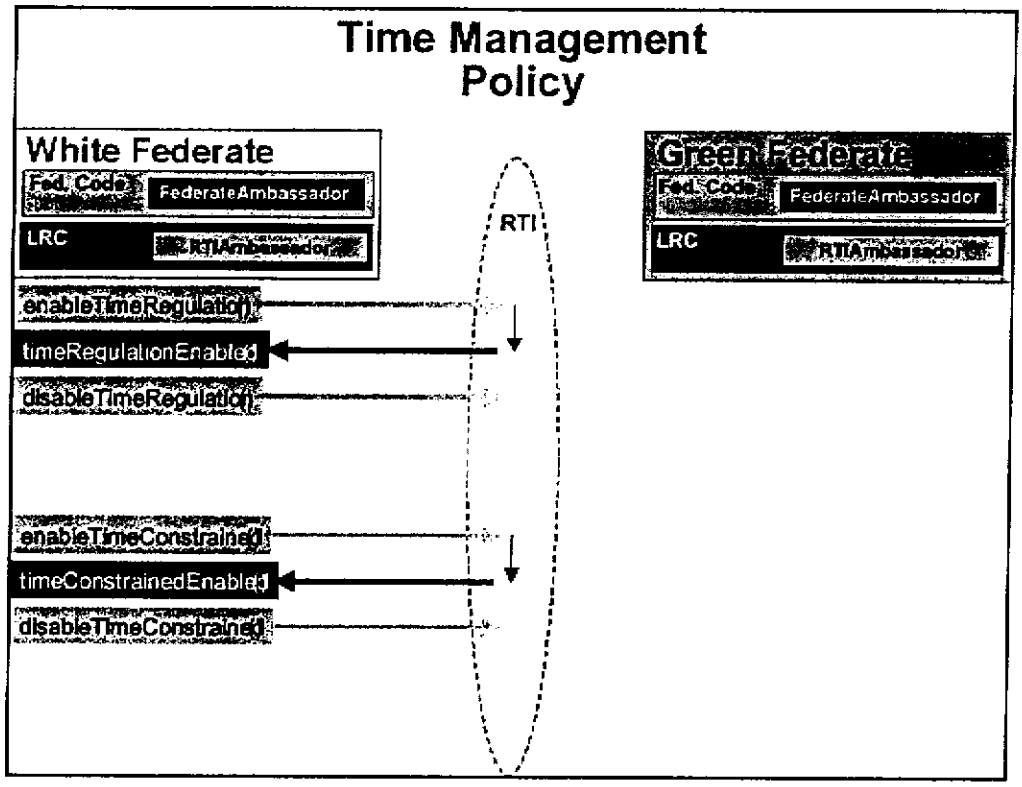

Time Management
Policy
White Federate
Fed. Code
FederateAmbassador
LRC
RTI
Green Federate
Fed. Code
FederateAmbassador
LRC
enableTimeRegulation()
timeRegulationEnabled()
disableTimeRegulation()
enableTimeConstrained()
timeConstrainedEnabled()
disableTimeConstrained()

圖 2-9

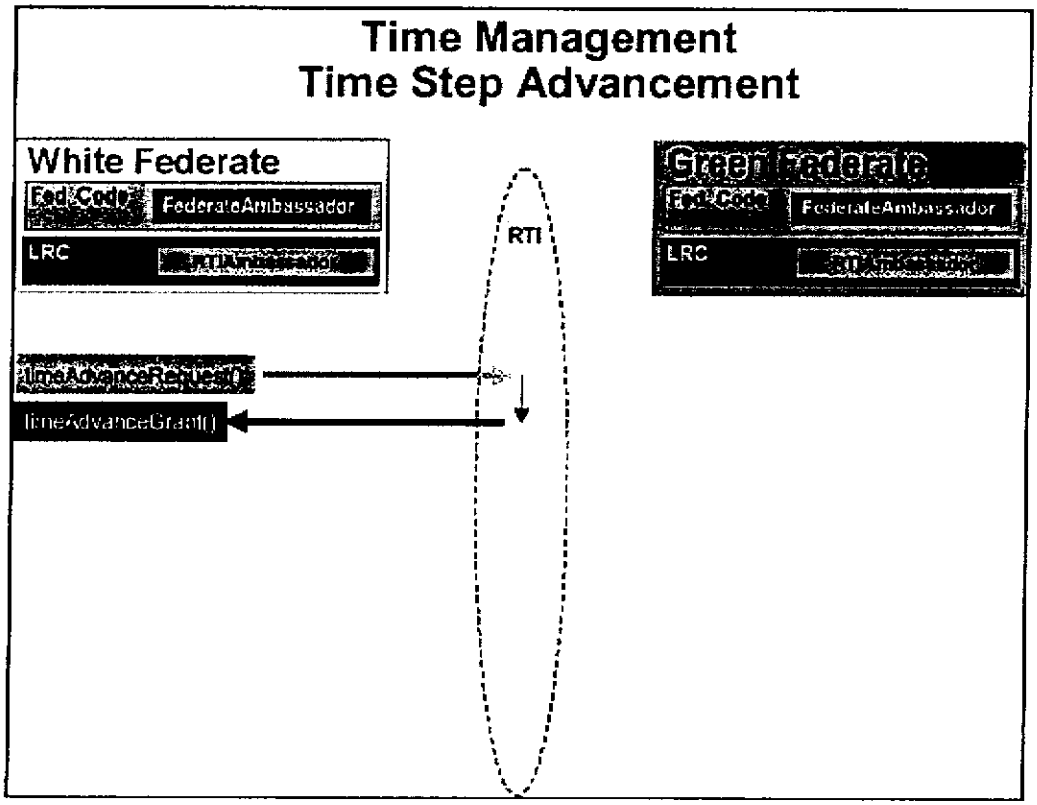

Time Management
Time Step Advancement
White Federate
Fed. Code
FederateAmbassador
LRC
RTI
Green Federate
Fed. Code
FederateAmbassador
LRC
timeAdvanceRequest()
timeAdvanceGrant()

圖 2-10

圖 2-10 為 Federate 以 Time Step 方式執行時，要求時間推進所需的介面函式。Federate 以 timeAdvanceRequest()來向 RTI 要求時間推進，當 RTI 認可時間推進時，以 timeAdvanceGrant()通知 federate 可以推進。

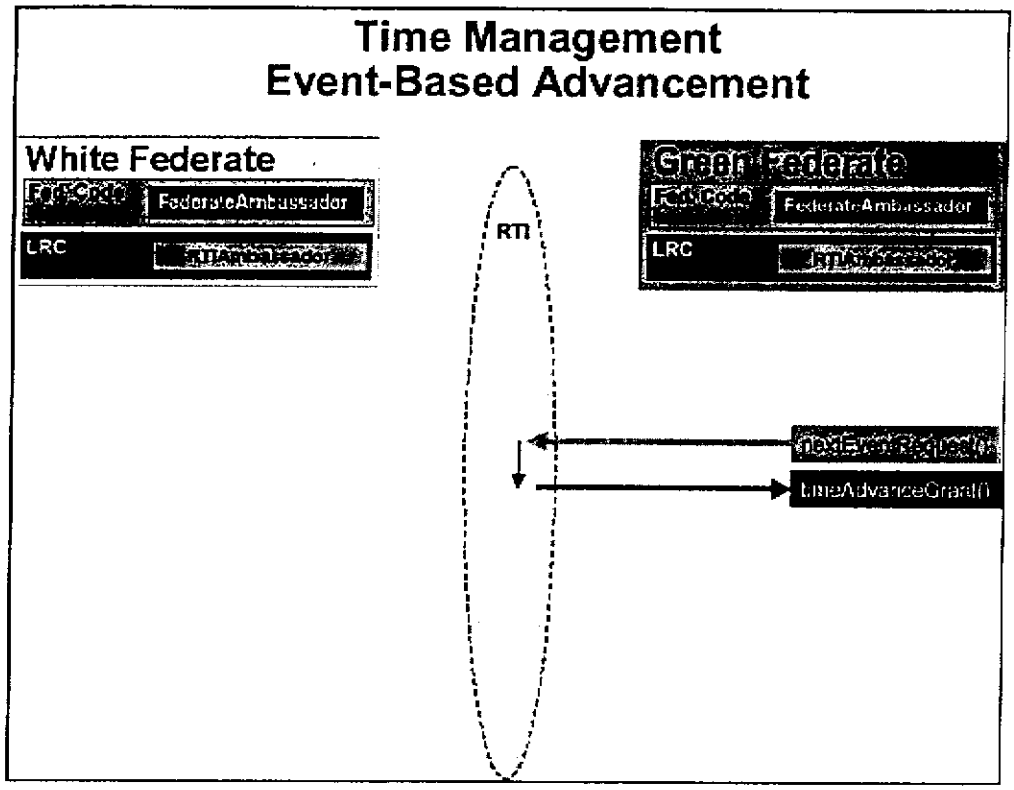


圖 2-11

圖 2-11 為 Federate 以 Event-based advance 方式執行時，要求時間推進所需的介面函式。Federate 以 nextEventRequest()來向 RTI 要求時間推進，當 RTI 認可時間推進時，以 timeAdvanceGrant()通知 federate 可以推進。

圖 2-12 為 Federate 是 Optimistic 型態時，時間推進所需用到的介面函式。Federate 以 flushQueueRequest()來向 RTI 要求時間推進，當 RTI 認可時間推進時，以 timeAdvanceGrant()通知 federate 可以推進。

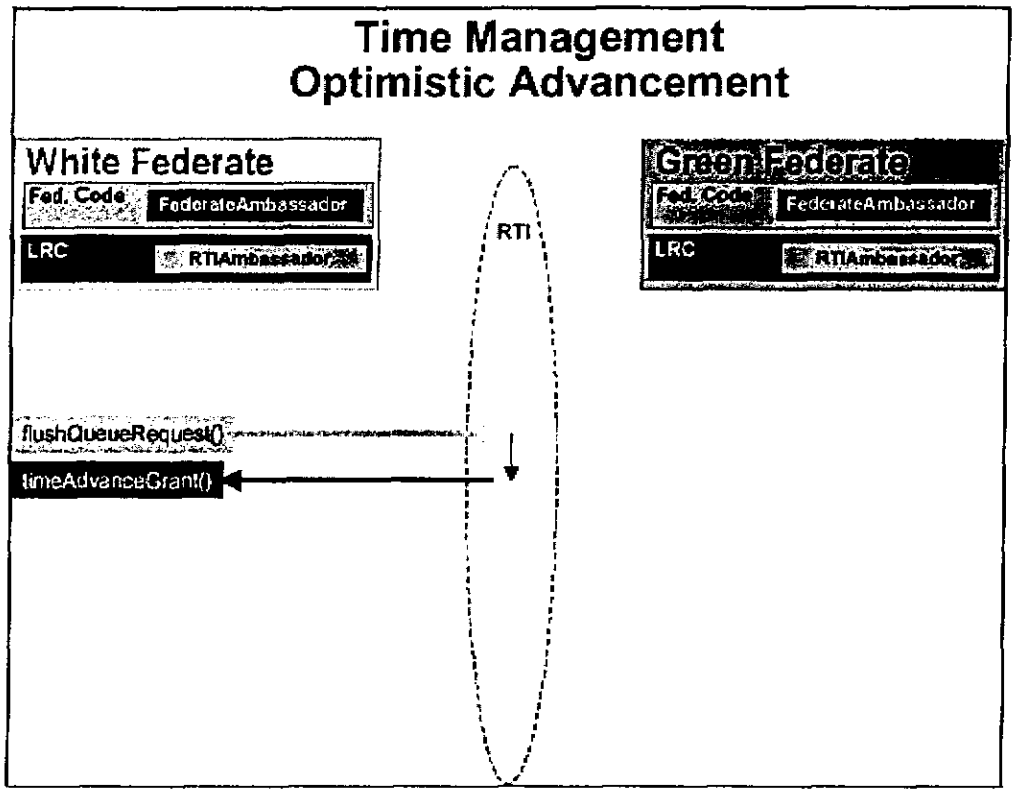


圖 2-12

圖 2-13 所列出的一些介面函式是提供 federate 針對校時，要求或設定 lookahead 之類所額外增加的工具型介面函式。在需要時才呼叫使用。

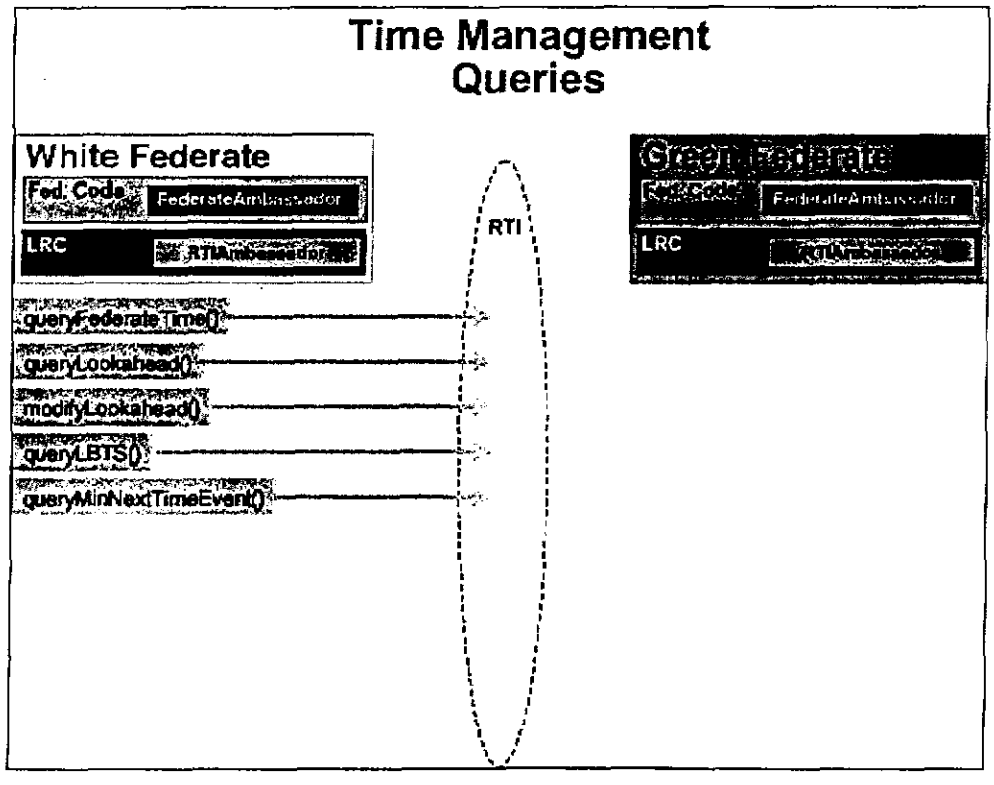


圖 2-13

C、Declaration Management

在 Declaration Management 的介面函式中，主要的目的是為了宣告 Object 及 Interaction 是屬於 Publish 或是 Subscribe。所以下面分成兩類說明：一是針對 Object，一是針對 Interaction。

在圖 2-14 中，federate 先以 getObjectClassHandle()來向 RTI 取得一個物件的 handle 值或物件名稱，再以此獲得的名稱運用 getAttributeHandle()來向 RTI 取得一個物件 instance 的 handle 值或名稱。接下來，就可以使用 publishObjectClass() 向 RTI 宣告此物件是 "Publish"屬性。同樣的，如果一個 federate 需要知道某些物件的屬性資料，就必須先以 subscribeObjectClassAttribute()來向 RTI 宣告。當 RTI 獲得這個宣告後，RTI 會以 startRegistrationForObjectClass()告知目標物件之 federate。 同樣的，當一個 federate 執行至某些階段，發現並不需要再接收到這些 subscribe 的物件屬性時，可以使用 unsubscribeObjectClass()通知 RTI，RTI 接到通知後，以 stopRegistrationForObjectClass()告知目標物件之 federate，讓此 federate 可以從名單中移除。另外，如果曾經宣告過 publish 的物件，若需要更改或放棄，則以 unpublishObjectClass()向 RTI 宣告。

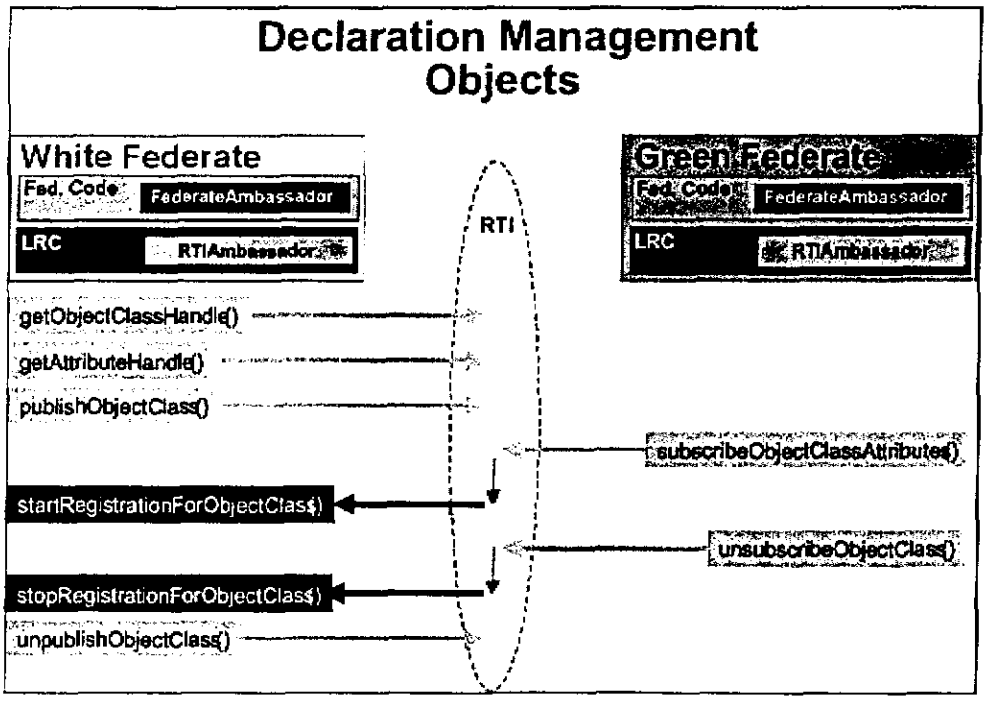


圖 2-14

圖 2-15 中，federate 先以 getInteractionClassHandle()來向 RTI 取得一個 Interaction class 的 handle 值或物件名稱。接下來，就可以使用 publishInteractionClass()向 RTI 宣告此 Interaction 是 "Publish"屬性。同樣的，如果一個 federate 需要接收的到這個 Interaction，就必須先以 subscribeInteractionClass()來向 RTI 宣告。當 RTI 獲得這個宣告後，RTI 會以 turnInteractionOn()告知目標 Interaction 所屬之 federate。同樣的，當一個 federate 執行至某些階段，發現並不需要再接收到這些 subscribe 的 Interaction，可以使用 unsubscribeInteractionClass()通知 RTI，RTI 接到通知後，以 turnInteractionOff()告知目標物件之 federate，讓此 federate 可以從名單中移除。另外，如果曾經宣告過 publish 的物件，若需要更改或放棄，則以 unpublishInteractionClass()向 RTI 宣告。

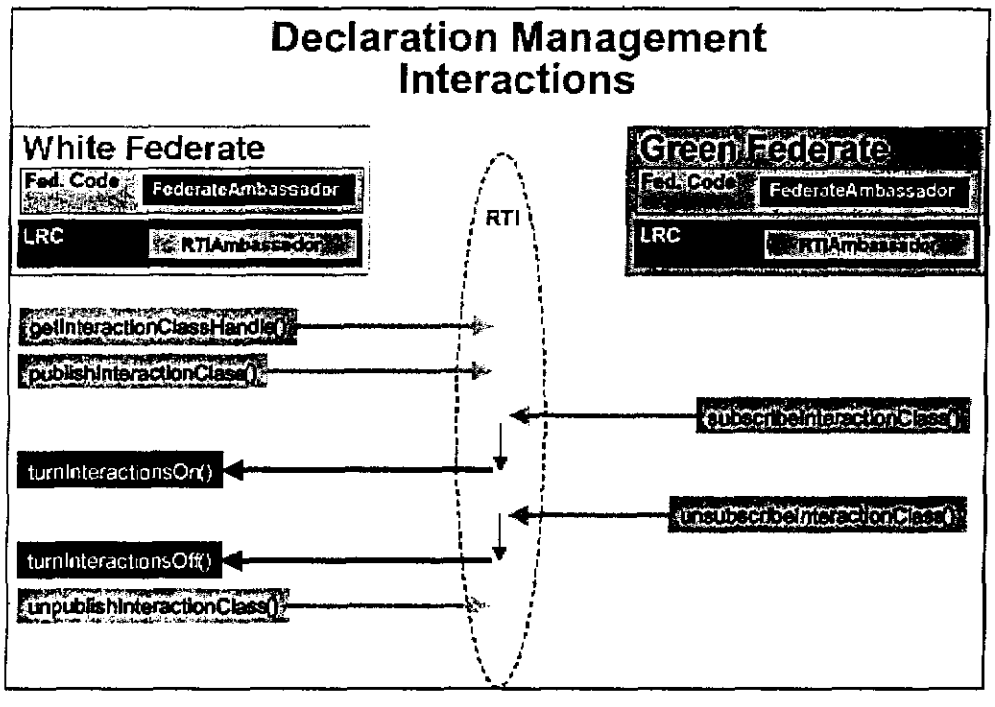


圖 2-15

D、Object management

Object Management 是由一群針對物件的啟始、增加、執行或移除功能所需要的介面函式所組成。

圖 2-16 為物件之 instance 的註冊、發現、刪除等功能的標準步驟。當 federate 需要加入一個新的 object Instance 時，運用 registerObjectInstance()來向 RTI 提出要求，而 RTI 在獲得要求後會以 discoverObjectInstance()通知所有曾經 Subscribe 這個物件的 Federates，告知 Instance 已經存在，接下來，RTI 還會以 turnUpdateOnForObjectInstance()來通知原 Instance 所屬 federate 可以開始執行並上傳資料。如果一個 Object 因為一些狀況必須移除，則 federate 以 deleteObjectInstance()通知 RTI，要求移除這個 Instance，RTI 收到後會以 removeObjectInstance()通知所有 subscribe 的 federates 移除這個 Instance。

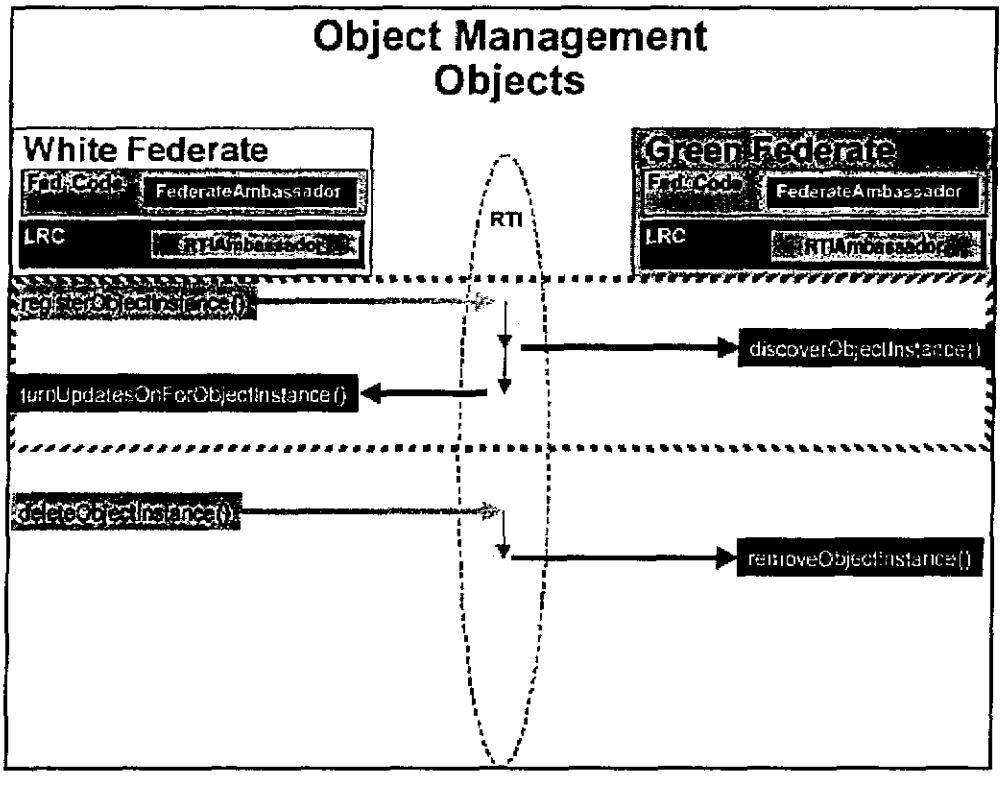


圖 2-16

圖 2-17 為 federation 執行時，物件資料的傳遞所須要之介面函式。當一個對某 Instance 之 attributes 宣告 subscribe 的 federate 需要資料的更新時，會以 requestClassAttributeValueUpdate()或以 requestObjectAttributeValueUpdate()來通知 RTI 送出要求，RTI 接到要求後會以 provideAttributeValueUpdate()通知目標 federate，此時，目標 federate 會以 updateAttributeValues()將新的資料告知 RTI，RTI 再以 reflectAttributeValue() call back 給提出要求的 federate，給予更新的資料。

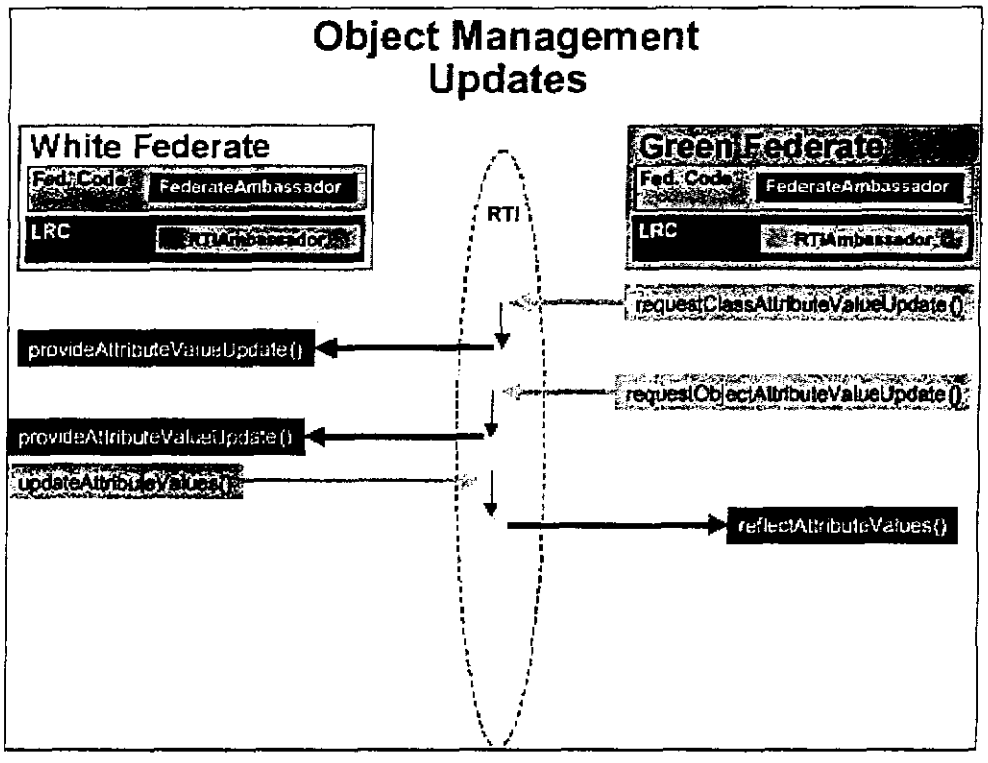


圖 2-17

圖 2-18 為 Interaction 的傳遞方式。如果一個 federate 要送出一個 interaction，會運用 sendInteraction()通知 RTI，RTI 在接到通知後，以 receiveInteraction()告知那些 subscribe 的 federates。

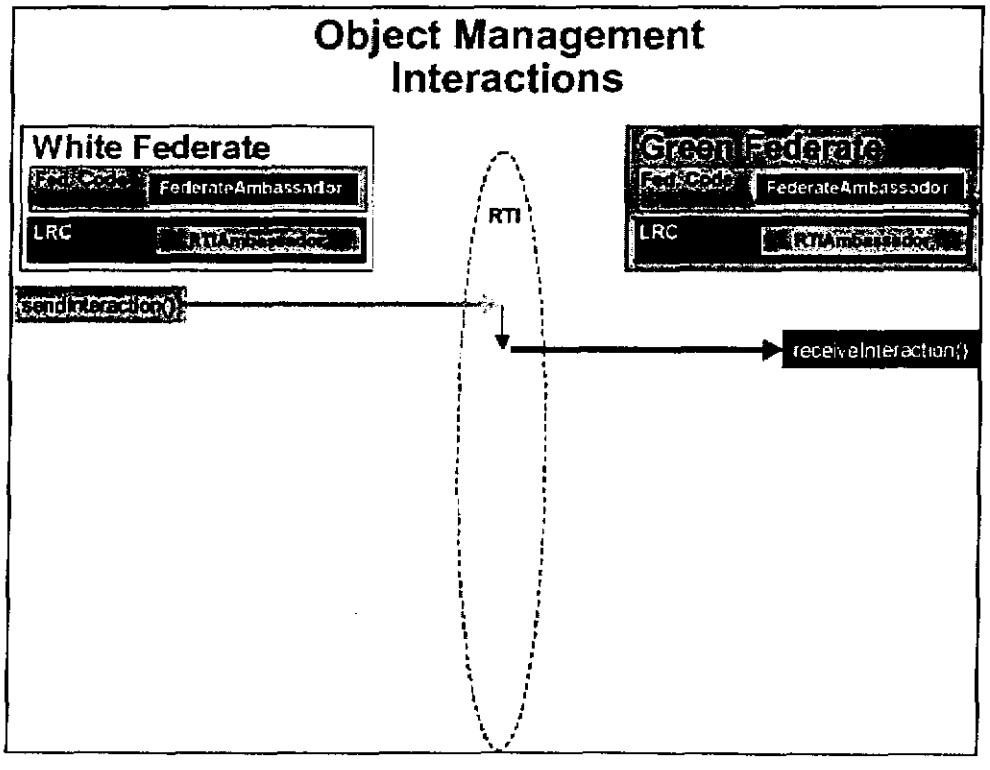


圖 2-18

E、Ownership Management

Ownership Management 是唯一的一個只在 execution 狀態時才用到的介面函式群組，目的在轉移 Object 的控制權。當然控制權的轉移也會牽動到其它 Management 的運作，因此在 Execution 狀態下，所有的 Management 都派的上用場。

圖 2-19 為物件所有權轉移方式為被動式的 Pull 時，所需要用到的介面函式。一個 federate 以 attributeOwnershipAcquisitionIfAvailable()向 RTI 要求一個物件的擁有權，RTI 會依當時此物件是否無任何 federate 擁有與否，以 attributeOwnershipUnavailable()或是 attributeOwnershipAcquisitionNotification()來 call back 告知沒有取得或成功取得物件的擁有權。

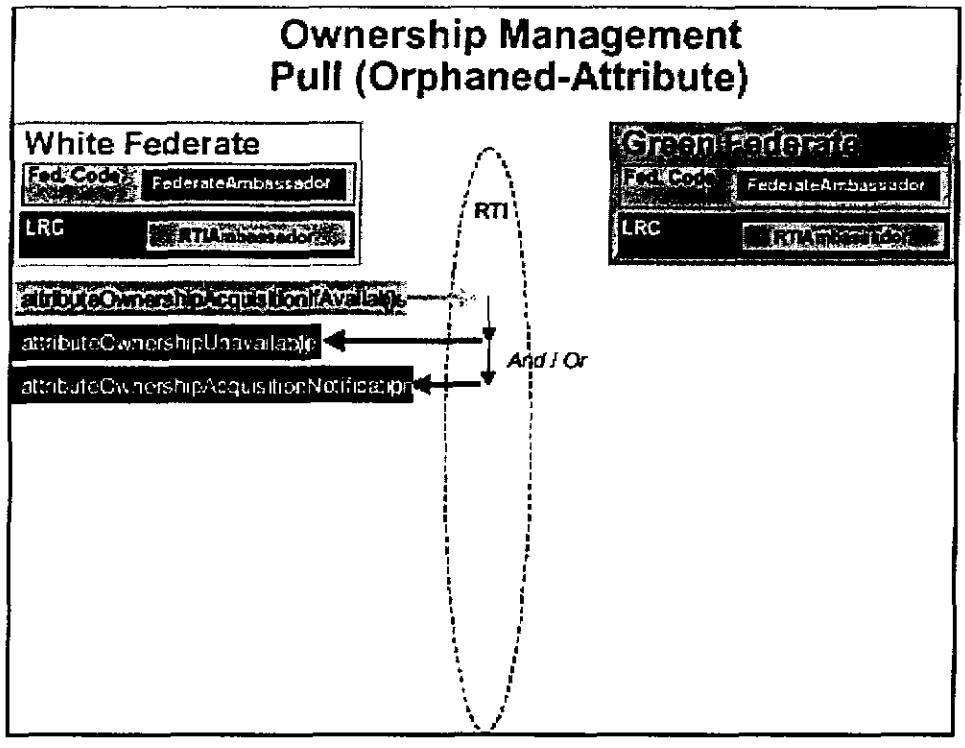


圖 2-19

圖 2-20 為物件所有權轉移方式為 Push 時，所需要用到的介面函式。一個 federate 以 negotiatedAttributeOwnershipDivestiture()向 RTI 提出放棄一個物件的擁有權，RTI 收到要求後會以 requestAttributeOwnershipAssumption()向其它的 Federates 提出詢問，若有任一個 federate 願意取得擁有權，則以

attributeOwnershipAcquisition() 告知 RTI，RTI 此時會同時以 attributeOwnershipDivestitureNotification()告知原來的 federate 以及願意取得擁有權之 federate 開始轉移擁有權。

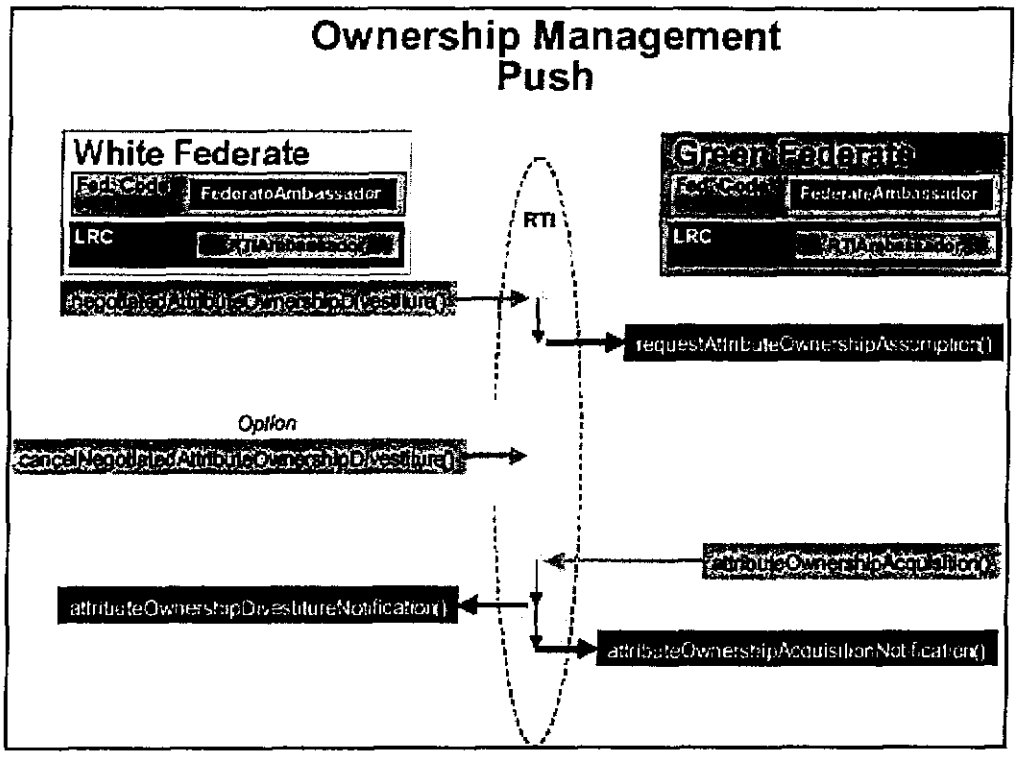


圖 2-20

圖 2-21 為物件所有權轉移方式為積極式的 Pull 時，所需要用到的介面函式。一個 federate 以 attributeOwnershipAcquisition ()向 RTI 要求一個物件的擁有權，RTI 會以 requestAttributeOwnershipRelease()向原擁有之 federate 提出要求，如果原擁有之 federate 想放棄擁有權，會以 attributeOwnershipReleaseResponse()來通知 RTI，RTI 再以 attributeOwnershipAcquisitionNotification()告知要求的 federate 已經取得物件擁有權。

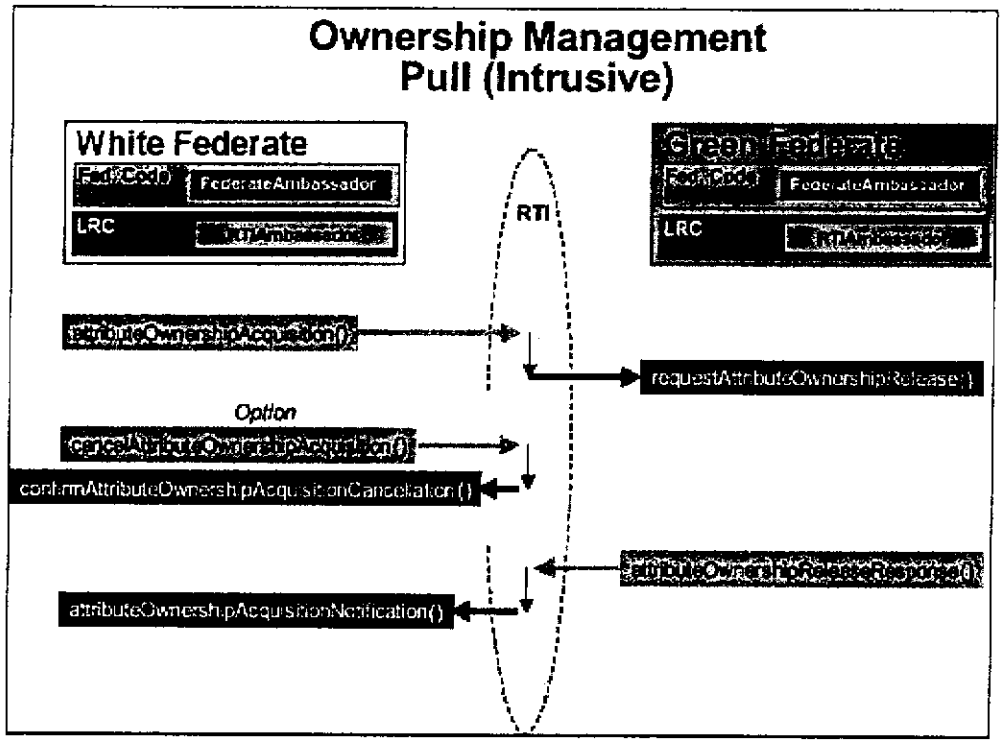


圖 2-21

F、Data Distribution Management

在 Data Distribution Management 中，主要的功能是以 "routing space"的機制區分 federates 的群組，在同一區域的 federates 才做資料的傳遞，以此達到提高 federation 效能的目的。因此，也會與 Object Management 產生微妙的關係。

圖 2-22 中顯示出如果要使用 DDM 的機制必須執行的第一步驟。Federate 以 createRegion() 來向 RTI 提出創造 routing space 的要求。或是以 notifyAboutRegionModification()來對 RTI 提出修改 space 的要求。當 federate 要離開前，以 deleteRegion()來要求 RTI 移除這個 Space。

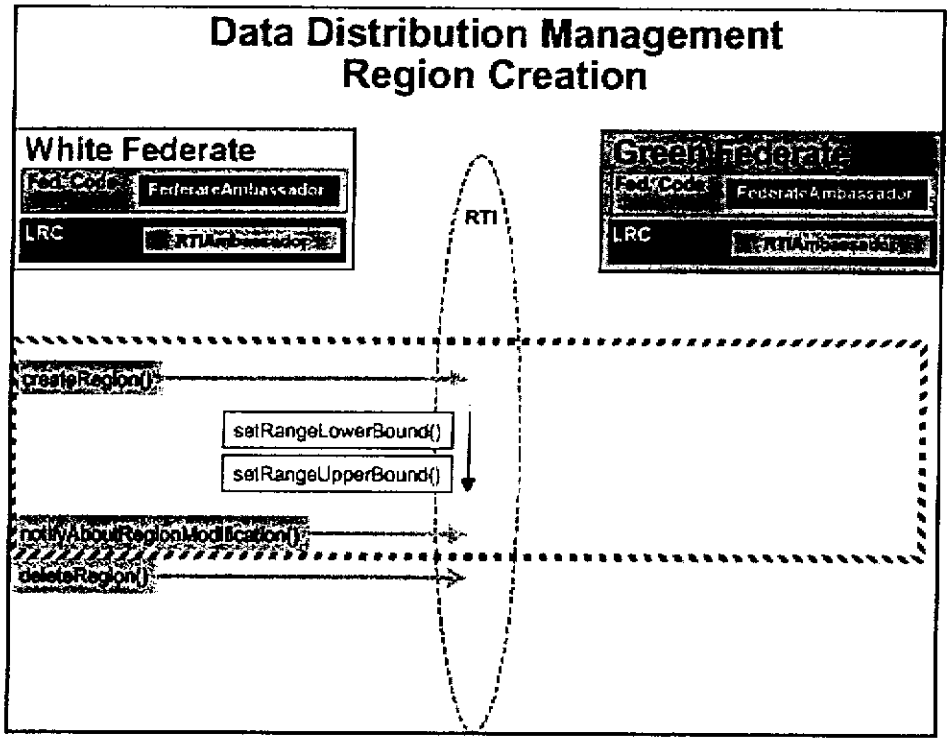


圖 2-22

圖 2-23、圖 2-24、圖 2-25 及圖 2-26 四張圖分別列出一個 federate 在 start-up 階段到 execution 階段所須使用到之介面函式。在這裡也提供了 Object Management 及 Declaration Management 中在 start-up 階段的相同功能之介面函式，也就是說，如果我們要使用 DDM 的機制，則某些 Object 宣告及註冊的介面函式就可以用圖中所示的介面函式取代。

在圖 2-23 中，先以 createRegion()創造 routing space，當 RTI 收到 routing space 的資訊後開始設定區塊，接著以 notifyAboutRegionModification()來修正物件與區域關係。在 fedearate 做完設定區域的動作後，就可以執行 Declaration Management 中的 publishObjectClass() 或 DDM 中的 subscribeObjectClassAttributeWithRegion()來宣告物件型態。在圖 2-24 中，以 enableAttributeRelevanceAdvisorySwitch()來打開更改 attribute 形態的開關，再以 registerObjectOnForObjectInstance()來註冊有區域宣告的物件實例，以利於進行接下來的一些步驟，如 discoverObjectInstance()。當模擬進入 execution 階段時，則以 requestClassAttributeValueUpdateWithRegion()來要求資料更新。在圖 2-25 中，則列出在 end 階段時，所需要使用的介面函式，以 unsubscribeObjectClassWithRegion()來移除宣告，接著以其它 management 的一

些介面函式進行標準的結束步驟，最後以 deleteRegion()來解除區域設定。而在圖 2-26 中，則列出在 execution 階段中，運作 interaction 傳遞時的標準方式，其中以 subscribeInteractionClassWithRegion()來宣告 interaction 的需要，並以區域畫分；以 sendInteractionWithRegion()依照區域來傳遞 interaction；最後，在 end 階段，透過 unsubscribeInteractionClassWithRegion()來移除區域的 interaction 宣告。

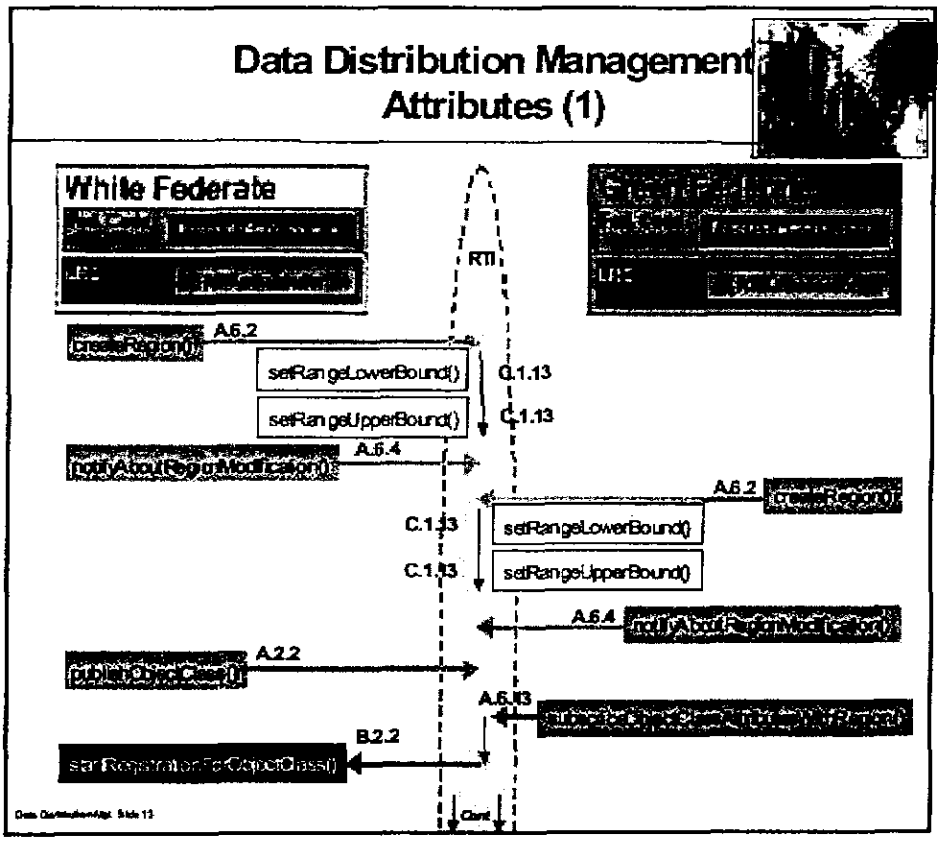


圖 2-23

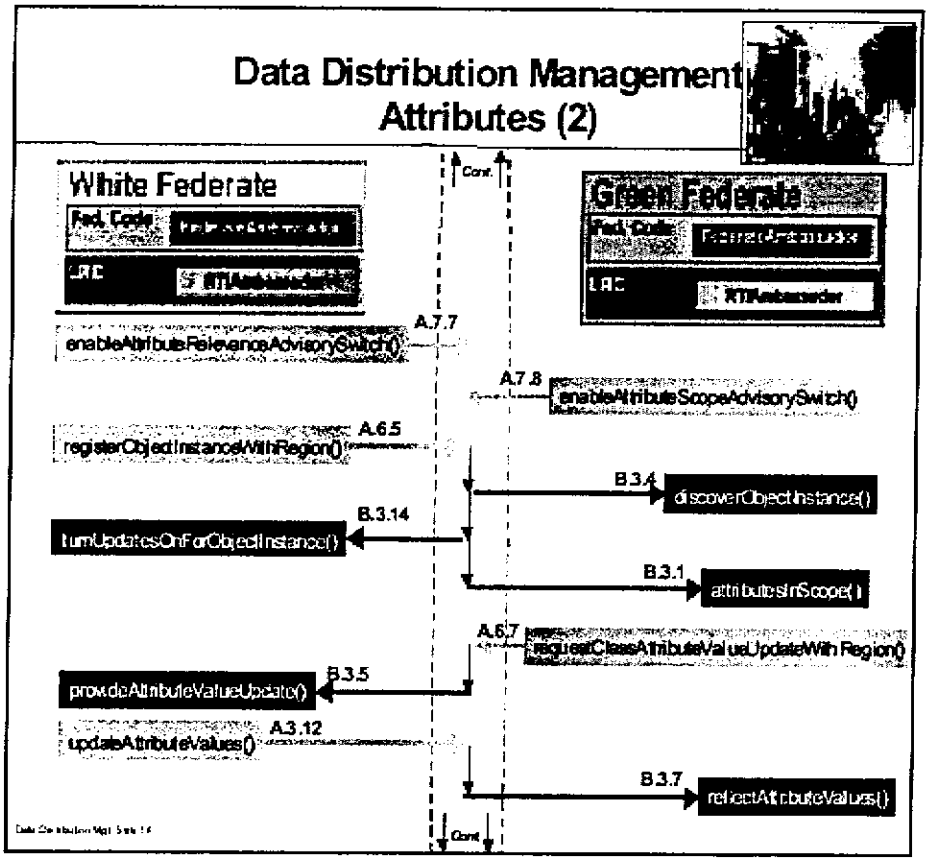

Data Distribution Management
Attributes (2)
White Federate
Green Federate
A.7.7
enableAttributeRelevanceAdvisorySwitch()
A.7.8
enableAttributeScopeAdvisorySwitch()
A.6.5
registerObjectInstanceWithRegion()
B.3.4
discoverObjectInstance()
B.3.14
turnUpdatesOnForObjectInstance()
B.3.1
attributeInScope()
A.6.7
requestClassAttributeValueUpdateWithRegion()
B.3.5
provideAttributeValueUpdate()
A.3.12
updateAttributeValues()
B.3.7
reflectAttributeValues()

圖 2-24

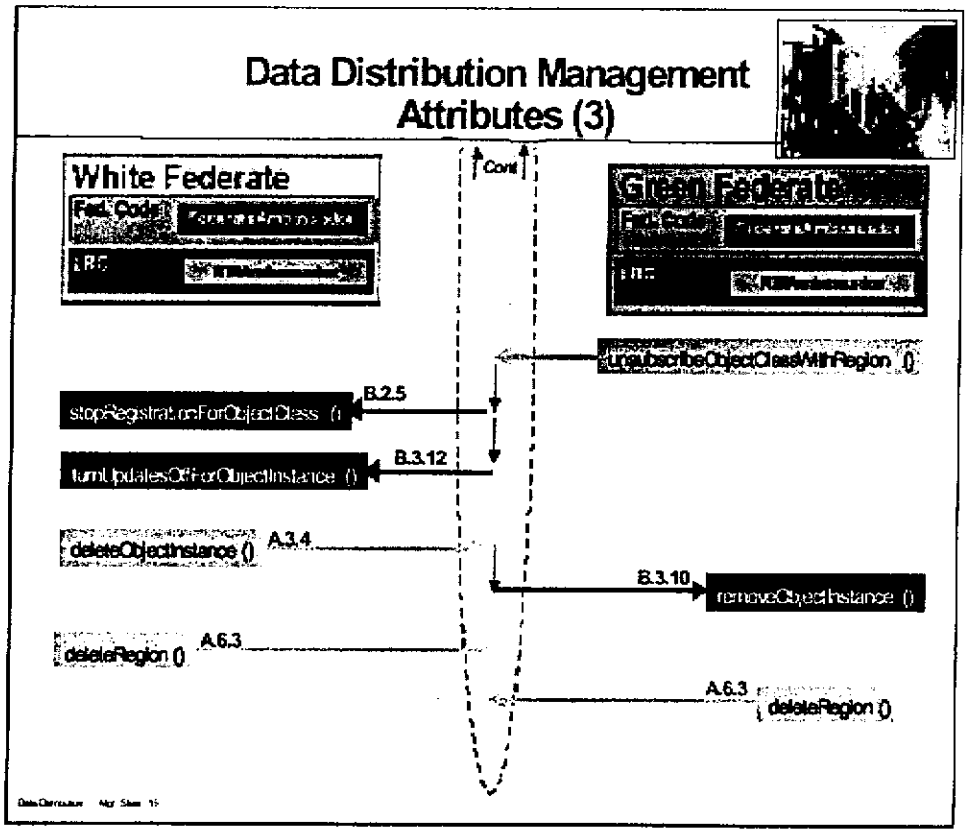

Data Distribution Management
Attributes (3)
White Federate
Green Federate
unsubscribeObjectClassWithRegion ()
B.2.5
stopRegistrationForObjectClass ()
B.3.12
turnUpdatesOffForObjectInstance ()
A.3.4
deleteObjectInstance ()
B.3.10
removeObjectInstance ()
A.6.3
deleteRegion ()
A.6.3
deleteRegion ()

圖 2-25

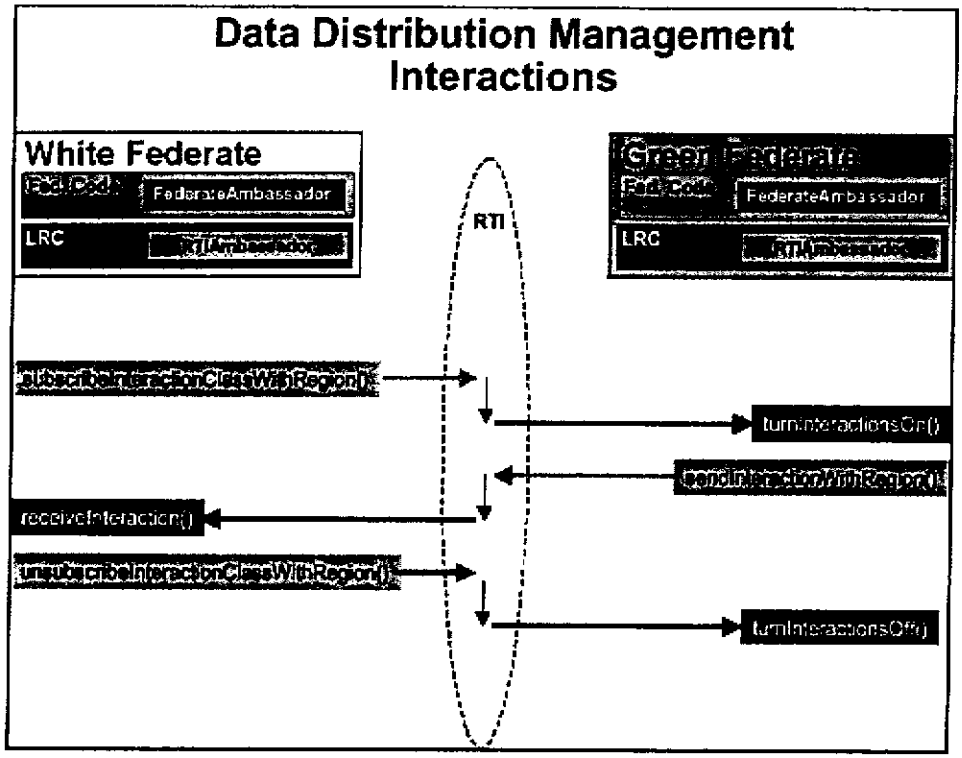

Data Distribution Management
Interactions
White Federate
FederateAmbassador
LRC
RTI
Green Federate
FederateAmbassador
LRC
turnInteractionsOn()
receiveInteraction()
turnInteractionsOff()

圖 2-26

第三章、RTI 程式庫分析方式

在本計劃中之程式庫分析流程大致可分為下列幾個步驟：

1. 依照界面名稱、界面類型、執行方式等依據，將所有 RTI 介面函式做一個 Top-Down 的分類，並以樹狀圖表示。再將分類之樹狀圖各節點命名，並做第一次的分類核對，判斷分類合理度，並做出修正。

2. 將樹狀圖末端各個介面函式中的參數列出，開始回溯分析，將相同的參數提升至前一層之節點，以此再度核對分類的合理性，並選出一些位於同一分支同時擁有共同參數的介面函式，可以包裝在一起，並自動化處理屬於內部之動作。

3. 依照前面的分析，配合 RTI 執行的狀態，歸納成 Class Library 的分析結果。

這三個步驟分述於以下三節：

3.1 分類原則

在分析的第一步驟，依照 HLA 的六大管理模組為基準，先將介面函式分成六大類，如下：

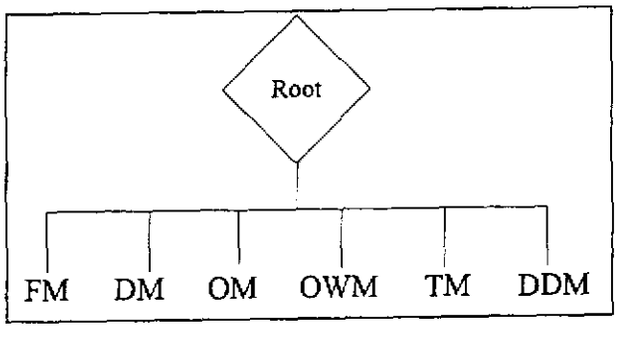


圖 3-1

其中：

1. FM：Federation Management

2. DM：Declaration Management

3. OM：Object Management

4. OWM：Ownership Management

5. TM：Time Management

6. DDM：Data Distribution Management

接下來，我們依照這六大分支開始各別分析如下：

A、Federation Management (FM)

在 Federation Management 中，我們依照這些介面函式 在執行時的功能按照 Programmer Guide 的說明將其再分為四大類。如下：

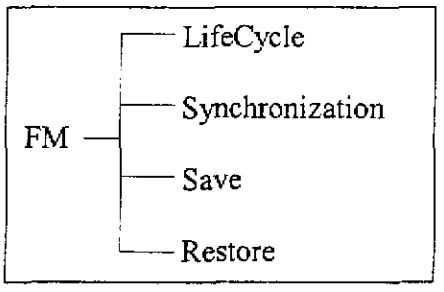


圖 3-2

a. Life Cycle 部份：以 Federation Management 的 Life cycle 處理為主，因為無需要再做更進一步的細分，所以本分支模組如下圖。

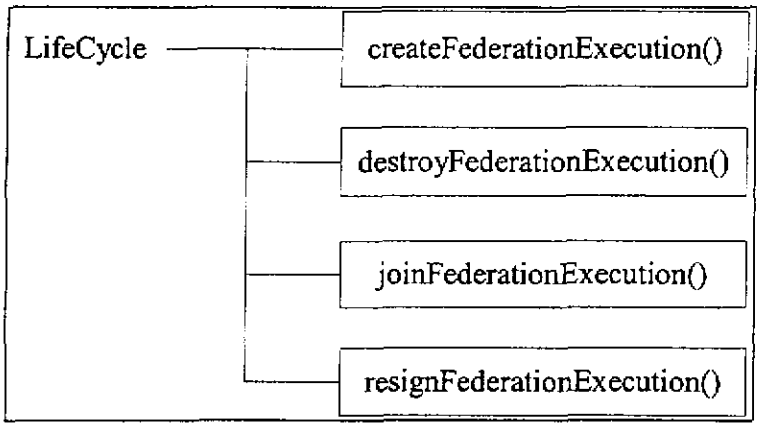


圖 3-3

b. Synchronization 部份：為針對 Federation 不同階段的同步處理為主，以執行時的即時性可再細分為兩類，一是 register_sp，一是 sp_achieve，如下圖所示。

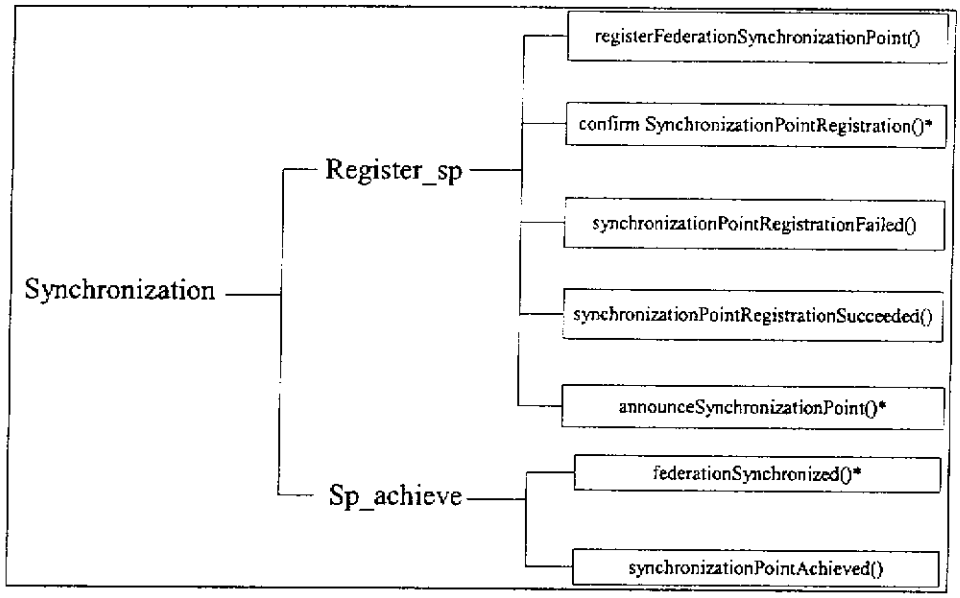


圖 3-4

c. Save 部份：包含的是當 Federation 儲存時的所需介面函式，這些介面函式在執行上已經無法再細分出類別，因此列為同一類型，如下圖所示。

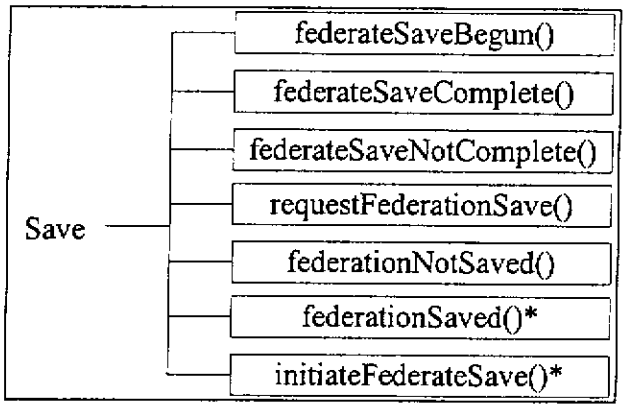


圖 3-5

d. Restore 部份：這個分支所包含的介面函式為與 federates restore 相關的所有介面函式，在執行時是同一個動作階段，因此無需再細分，如下圖所示。

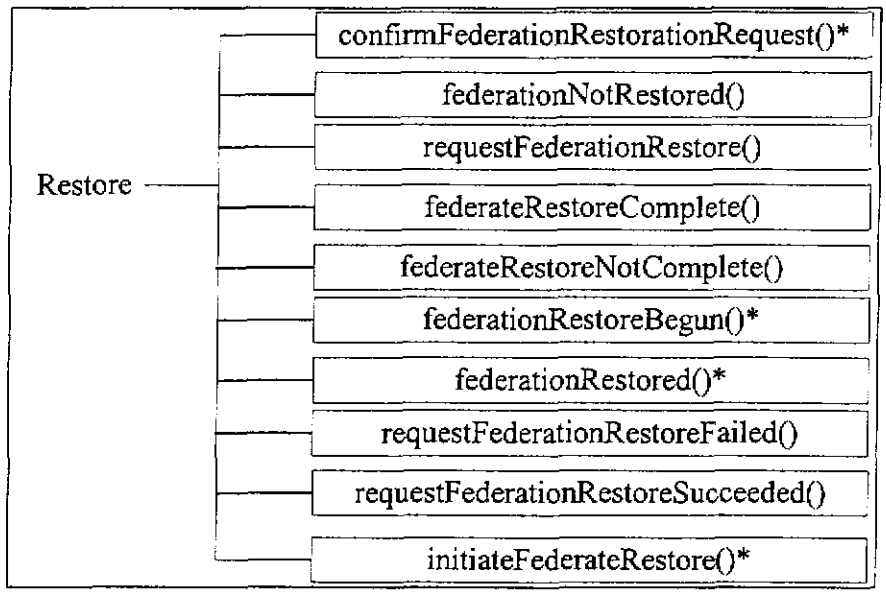


圖 3-6

B、Declaration Management(DM)

這個分支是 Declaration Management 相關的界面，可以依照對象再細分為：Object 以及 Interaction，如下圖所示。

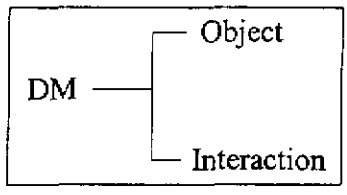


圖 3-7

a. Object 部份：Federate 對 RTI 做出 Object 的宣告，可以再以細部執行動作分為 Publish、Subscribe 及 Registration 三類，如下圖所示。

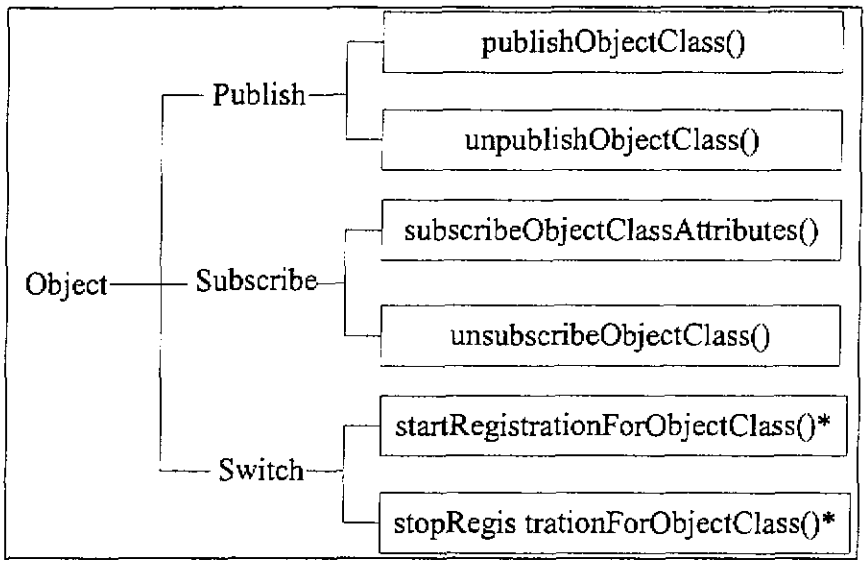


圖 3-8

b. Interaction 部份：Federate 對 RTI 做出 Interaction 的宣告，可以再以細部執行動作分為 Publish、Subscribe 及 Turn On&Off 三類，如下圖所示。

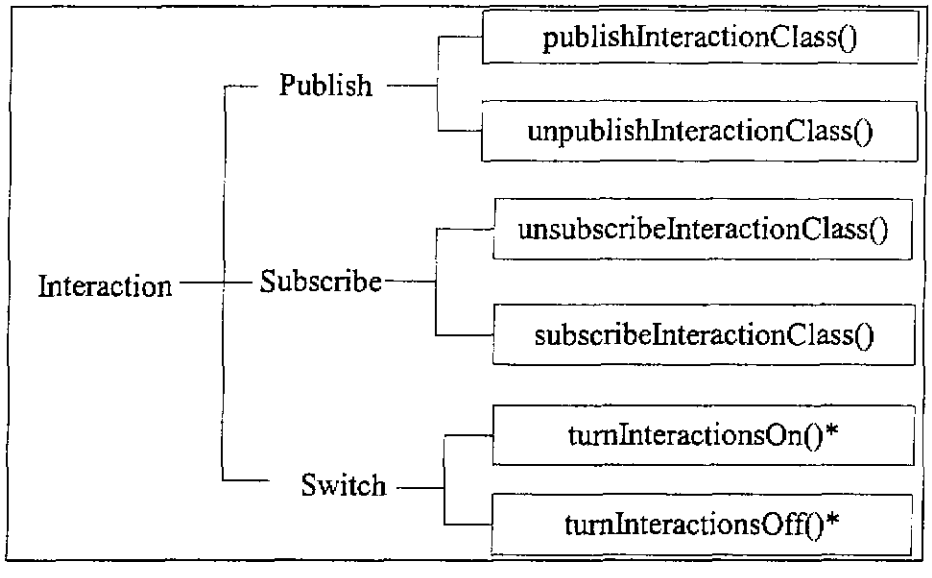


圖 3-9

C、Object Management(OM)

這個分支，針對的為 Object Management 的界面群，以對象為法則再細分為：Instance、Attribute 及 Interaction 三類。如下圖所示。

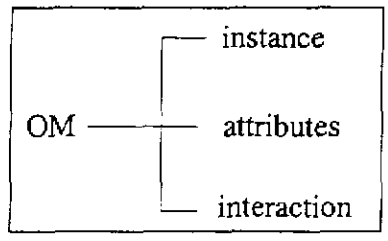


圖 3-10

a. Instance 部份：針對物件的 Instance 處理，以執行之動作來再細分為，Object Instance Register/Discover、Object Instance Delete 以及 Object Instance Turn On/Off 三類。如下圖所示。

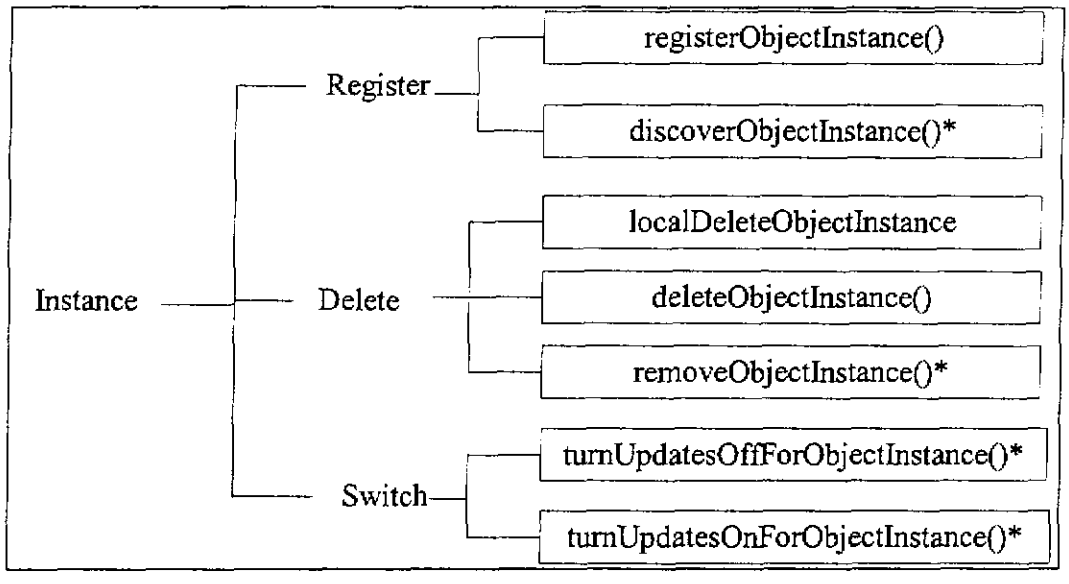


圖 3-11

b. Attribute 部份：將 Object Management 界面中，針對將 Instance Attribute 為執行對象的介面再分為，Active 狀態及 Passive 狀態兩種。如下圖所示。

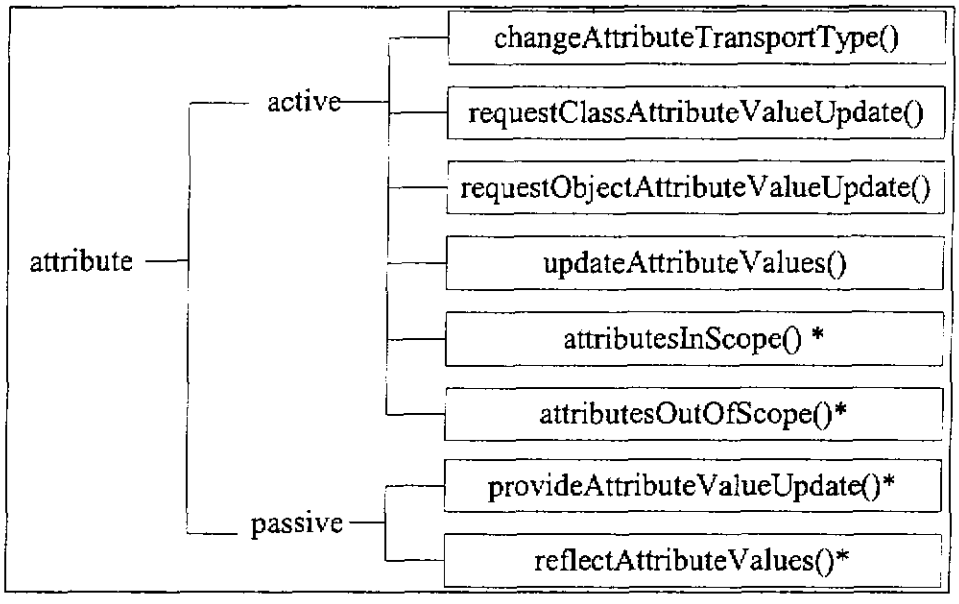


圖 3-12

c. Interaction 部份：此部份為 Object Management 界面群中，以 Interaction 為對象的一群。如下圖所示。

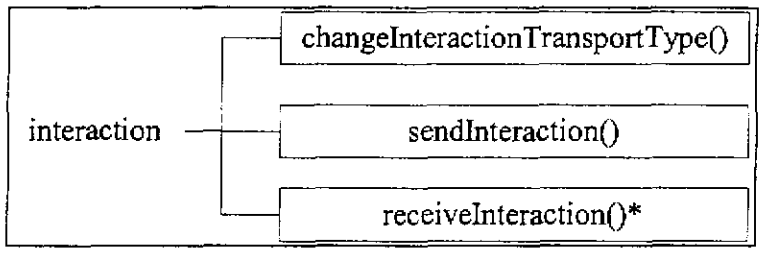


圖 3-13

D、Ownership Management(OWM)

這個分支為針對 Federation 在執行階段中，物件所有權轉移功能的界面群。可以再以執行方式或功能細分為：Pull 方式執行、Push 方式執行、Unobtrusive Acquisition 方式執行以及擁有權 Query 四大類。如下圖所示。

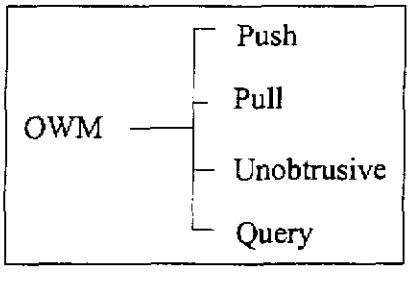


圖 3-14

其中：

- Push：Ownership Management Push
- Pull：Ownership Management Pull
- Unobtrusive：Ownership Management Unobtrusive Acquisition
- Query：Ownership Management Query

a. Push 部份：已擁有之 Federate 以主動方式讓出物件中 Instance 所有權時所需要用到的界面組成。

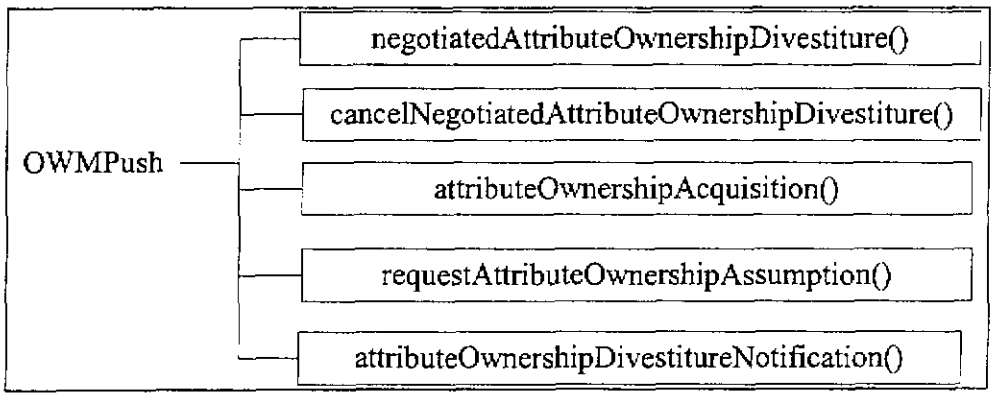


圖 3-15

b. Pull 部份：希望擁有之 Federate 以提出要求的方式取得物件中 Instance 所有權時所需要用到的界面組成。

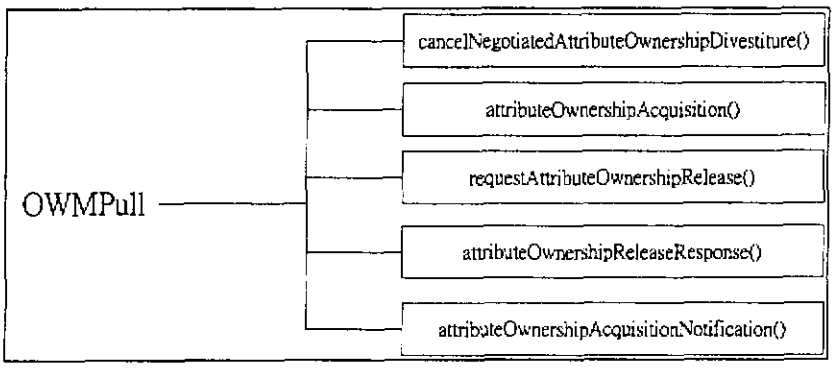


圖 3-16

c. Unobtrusive：希望擁有之 Federate 以協商的方式來取得物件中 Instance 所有權時所需要用到的界面組成。

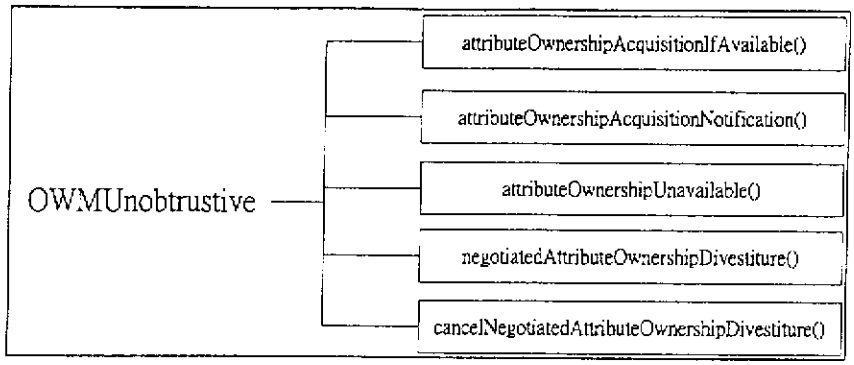


圖 3-17

d. Query 部份：Federate 查詢擁有權歸屬或宣告擁有權歸屬需要使用的界面群組。

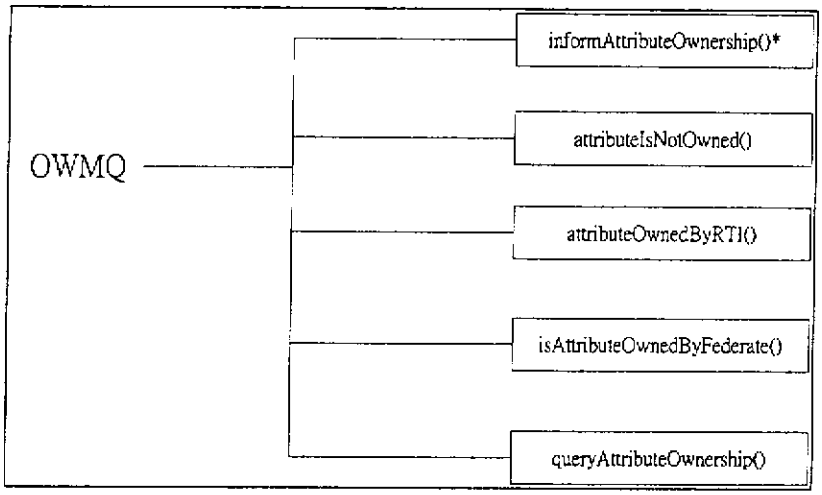


圖 3-18

E、Time Management(TM)

這個分支當中，所有的界面都是用來處理 Federation 的同步問題，所以一開始我們直接以處理的對象來分為兩大類，一為針對 Federation，另一為針對 Object。如下圖：

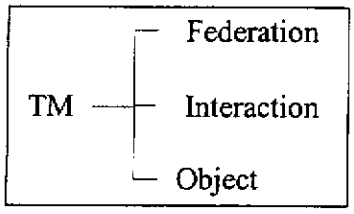


圖 3-19

a. Federation 部份：此部份為整個 Federation 在執行時，由起始到模擬進入執行階段，直到結束，所需要的一些同步處理界面。依照執行的狀態不同，又分為下面幾種。

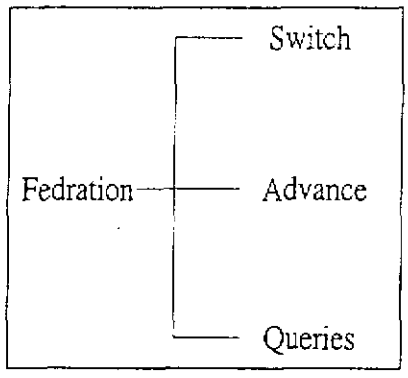


圖 3-20

Switch 節點的功能在於設定 federate 的時間關係屬性，是以 regulation 來主動要求推進時間，或是以 constrained 的被動方式推進時間。

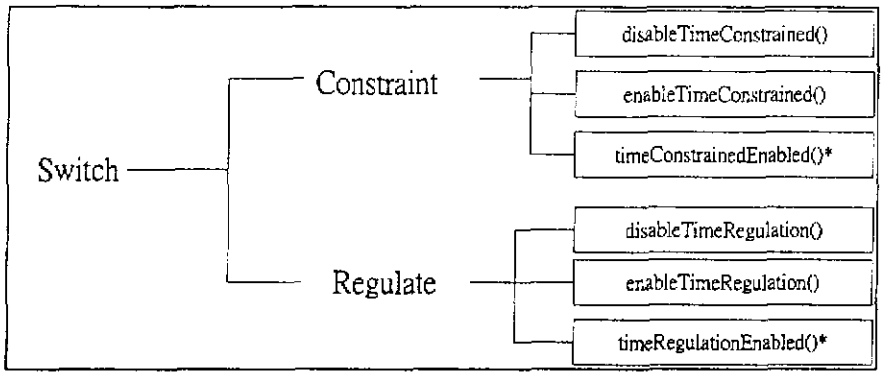


圖 3-21

其中：

- Constraint：Time Management Federate Switch Time Constrained
- Regulate：Time Management Federate Switch Time Regulation

在模擬執行時，推進的模式，可以分為：Step 方式(固定時間間距推進)、event 方式(以 event 來驅動時間的前進)、Optimistic (一種特例狀況的模擬限制)。

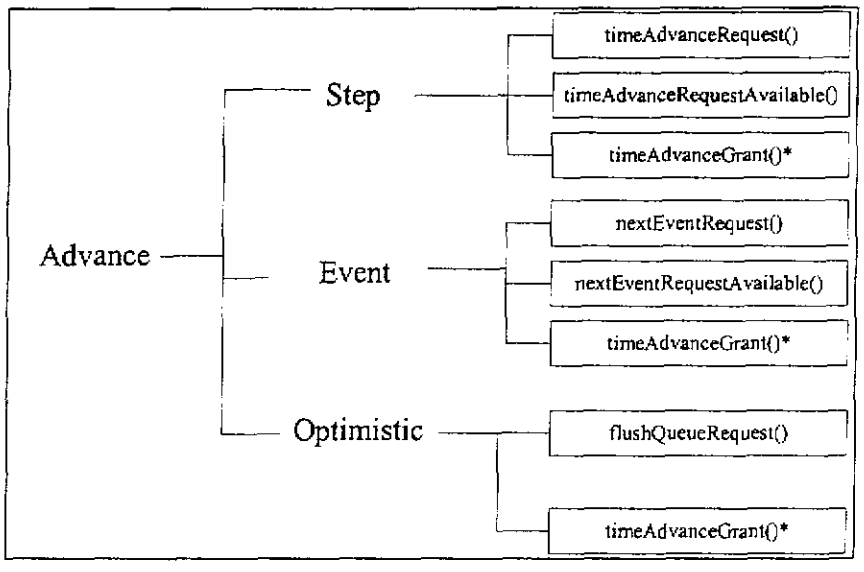


圖 3-22

Query 部份為 time management 當中的工具模組，用來取得或更改一些時間數值(如：LBTS 或 Lookahead)，以利於各個 federate 的運作以及時間的同步。

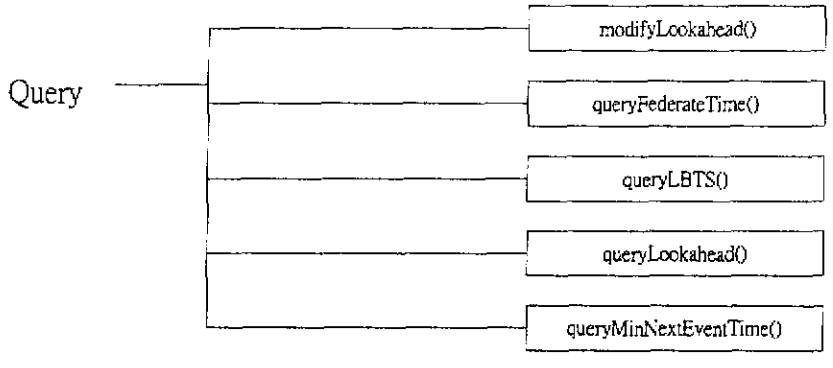


圖 3-23

b. Object 部份：此部份為整個 Federation 在執行時，針對物件之 Attribute 的同步關係設定(屬於主動或被動)及物件本身之時間標記設定之界面群。

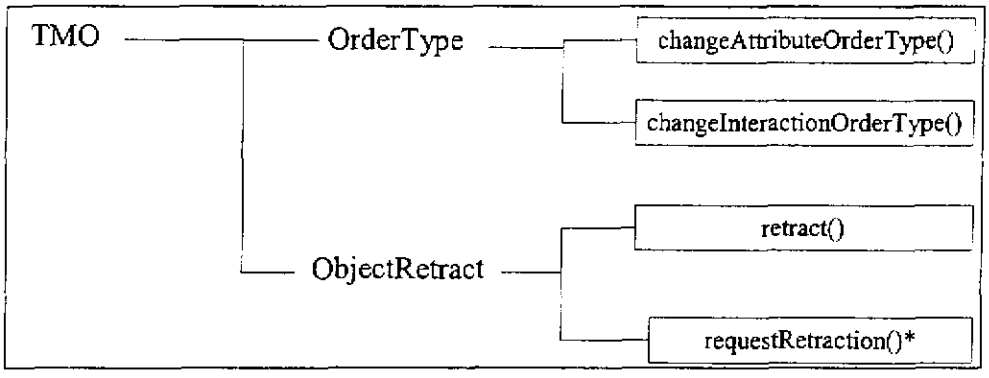


圖 3-24

F、Data Distribution Management(DDM)

這個分支當中，界面群主要是在處理區域管理(或資料分散管理)，以其執行的對象及狀態可以分為下圖中的三類。

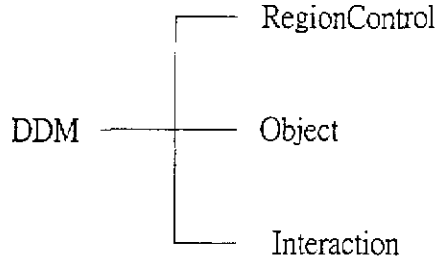


圖 3-25

a. RegionControl 部份：這個部份的功能在處理 routing space 的設定及修改，如下圖。

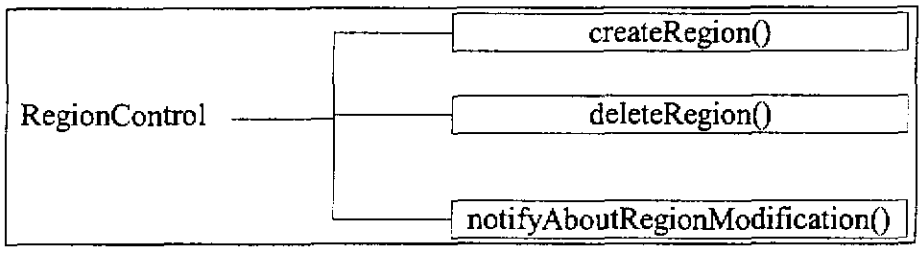


圖 3-26

b. Object 部份：這個部份是處理 Object 與區域的關係設定以及 Object 的宣告。

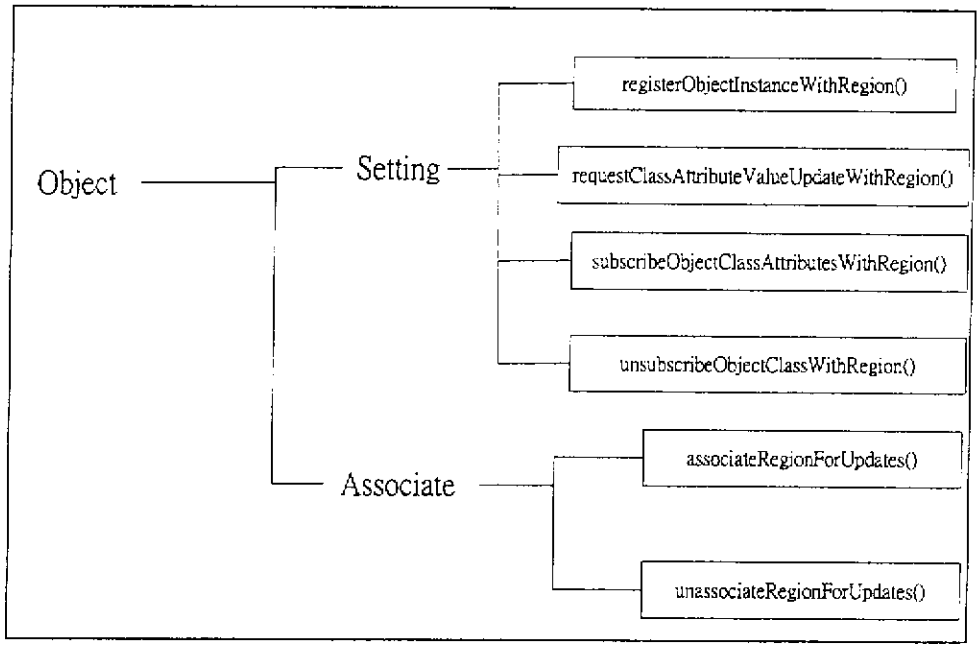


圖 3-27

c. Interaction 部份：這個部份是處理 Interaction 與區域的關係設定以及 Interaction 之宣告。

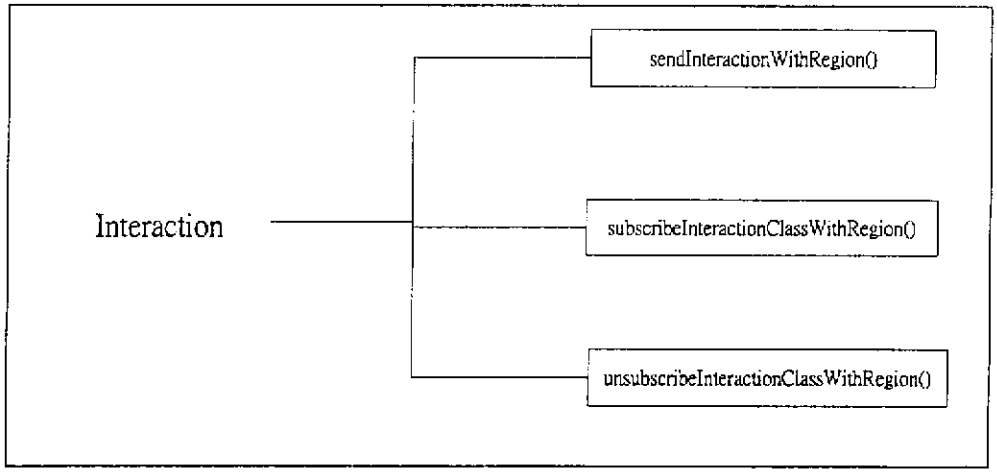


圖 3-28

3.2 分析屬性以及判斷自動化

依據前一節之分類結果，我們開始分析其內部的參數，並將同一節點下相同的參數往上提一層，希望能找出共用的 I/O，在包裝時可以做為是否自動化的準則。(Self 欄位所列出的為各個介面函式中的 attributes。相同的 attribute 會被提升至前一個節點，不保留於 Self 當中；但如果提升層級超過第二層則以箭頭標示，並保留第一提升層級內之 attribute。)

A、Federation Management 部份

表 3-1 部份，為 LifeCycle 節點與其所屬之介面函式的 attribute 比較表，由這張表我們可以得到一個共同的 attribute 為 executionName，所以如果 lifeCycle 節點有可能被包裝成一個 class 的話，則 executionName 為其共同的 input 值，而位於 Self 內的其它 attribute 則為內部參數。

表 3-2 部份，為圖 3-4 部份的 Synchronization 節點及其子節點 Register_sp、Sp_achieve 與其下介面函式所組成，可以被提出 Label 這個 attribute，當成共同的 Input 值。

表 3-3 部份為 Save 節點與其所屬之介面函式的 attributes，因為這個部份的 attribute 比較特殊，功能上是屬於可被區分為內部作業的一些參數，所以並不須要因為 attribute 的相同性，要求節點部份再做細分。

表 3-4 為 Restore 節點所屬的 attributes，其中的 saved label 為共同也是必須的輸入 attribute，所以提出至上一層。

	LifeCycle	Self
createFederationExecution()	executionName	FED
destroyFederationExecution()		
joinFederationExecution()		yourName FederateAmbassadorReference
resignFederationExecution()		theAction

表 3-1

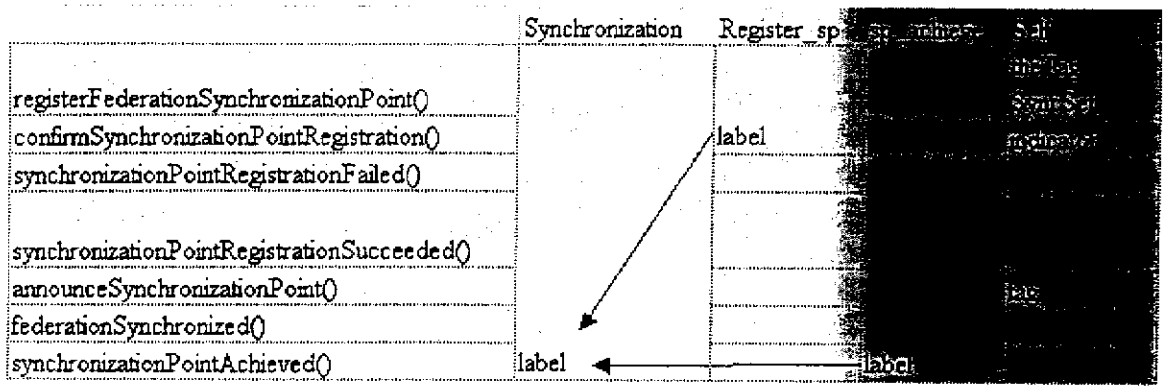


表 3-2

	Self
federateSaveBegun()	
federateSaveComplete()	save-success indicator
federateSaveNotComplete()	
requestFederationSave()	FederationSaveLabel Optional value of federation time
federationNotSaved()	
federationSaved()	save-success indicator
initiateFederateSave()	FederationSaveLabel

表 3-3

	FMRS	Self
confirmFederationRegistorationRequest()		indicator
federateNotRestored()		
requestFederationRestore()		
federateRestoreComplete()		
federateRestoreNotComplete()		
federateRestoreBegun()		
federateRestored()		
requestFederationRestoreFailed()		
requestFederationRestoreSucceeded()		
initiateFederateRestore()	Savedlabel	handle

表 3-4

B、Declaration Management

表 3-5 為 Declaration Management 當中的 Object 節點以及其子節點與介面函式的 attributes。其中，三個子節點 publish、subscribe、register 都可以由其所屬的介面函式中提出 "the Class"這個 attribute，因此我們可以試著再向前提一層至 Object 節點，以提供後續分析時評估包裝的層級。

(子節點)		Object	publish	subscribe	register	Self
publish	publishObjectClass()					attribute
	unpublishObjectClass()		theClass			
subscribe	subscribeObjectClassAttributes()					attribute active
	unsubscribeObjectClass()			theClass		
register	startRegistrationForObjectClass()					
	stopRegistrationForObjectClass()	theClass			theClass	

表 3-5

表 3-6 為 Declaration Management 當中的 Interaction 節點以及其子節點與介面函式的 attributes。其中，三個子節點 publish、subscribe、turn 分別可以提出相同的 attribute，而這三個 attribute 指得為同一個參數 theClass，只是 RTI 可以允許以 name 或 handle 來傳遞。

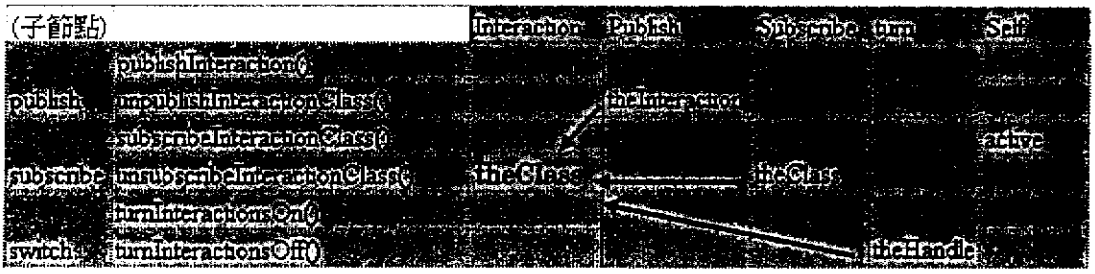


表 3-6

C、Object Management

表 3-7 為 Object Management 中的第一層節點 Instance 下的子節點與 attributes 列表。三個子節點分別為 register、delete、switch，各個節點都擁有一些提昇的 attributes，可以提供包裝時的考量。其中，由黑色框線框起的兩個介面函式為工具函式。

表 3-8 為 Object Management 中的第一層節點 "attribute"下的子節點與其 attributes 列表。兩個子節點分別為 active、passive，如果以功能來區分在 active 節點下的介面函式，可以用成對的介面函式來提昇 attribute，因此，如表中所列，被提升的 attributes 是可以被當成共同的內部參數處理。

表 3-9 為 interaction 節點下的介面函式之 attribute 狀況。其中，changeInteractionTransportType()是一個獨立的處理工具，因此，將另外兩個介面函式的共同 attributes 提升。

(子節點)		OMIO	OMID	OMIS	Self
Register	registerObjectInstance()	theObject			theClass
	discoverObjectInstance()				theObjectClass theObjectName
	getObjectInstanceName()				theHandle
	getObjectInstanceHandle()				theName
Delete	localDeleteObjectInstance()				theObject
	deleteObjectInstance()		objectHandle		
	removeObjectInstance()		theTime theTag		
Switch	turnUpdatesOffObjectInstance()			theObject	
	turnUpdatesOnObjectInstance()			theAttribute	

表 3-7

(子節點)		active	passive	Self
active	changeAttributeTransportType()			theObject theAttribute theType
	requestClassAttributeValueUpdate()			theClass theAttributes
	requestObjectAttributeValueUpdate()			
	updateAttributeValues()	theObject theAttribute		theObject theAttributes
	attributesInScope()	theObject		
	attributesOutScope()	theAttributes		
passive	provideAttibuteValueUpdate()			theObject theAttributes
	reflectAttributeValues()			ObjectHandle theAttribute theTag theTime theHandle

表 3-8

	interaction	Self
changeInteractionTransport()		theClass theType
sendInteraction()	theInteraction	
recieveInteraction()	theParameters theTime theTag	

表 3-9

D、Ownership Management

表 3-10 中，列出 Ownership Management 中 Push 節點(Push 方式轉移)的介面函式，其共同的 attribute 為 theObject。表 3-11，列出 Ownership Management 中 Pull 節點(Pull 方式轉移)的介面函式，其共同的 attribute 為 theObject。表 3-11，列出 Ownership Management 中 Unobtrusive 節點(Unobtrusive Acquisition 情況下轉移)的介面函式，其共同的 attribute 為 theObject。所以在表 3-10、3-11、3-12 中，我們發現都有共同的一個 attribute 為 theObject，可以提供後續分析時的參考。表 3-13 為 Ownership Management 的 Attribute query 相關工具，共同的 attribute 依然為 theObject，因此在內部被單獨呼叫使用時，可以更方便的使用共同參數。

	OWMPush	Self
negotiatedAttributeOwnershipDivestiture()		theAttributes theTag
cancelNegotiatedAttributeOwnershipDivestiture()		theAttributes
attributeOwnershipAcquisition()		theTag desiredAttributes
requestAttributeOwnershipAssumption()		theTag offeredAttributes
attributeOwnershipDivestitureNotification()	theObject	securedAttribute

表 3-10

	OWMPull	Self
confirmAttributeOwnershipAcquisitionCancellation()		theAttributes
cancelNegotiatedAttributeOwnershipDivestiture()		theAttributes
attributeOwnershipAcqisition()		theTag desiredAttributes
requestAttributeOwnershipRelease()		theTag candidateAttributes
attributeOwnershipReleaseResponse()		theAttributes
attributeOwnershipAcquisitionNotification()	theObject	securedAttribute

表 3-11

	OWMUnAc	Self
attributeOwnershipAcquisitionIfAvailable()		desiredAttributes
attributeOwnershipAcquisitionNotification()		securedAttribute
attributeOwnershipUnavailable()		desiredAttributes
negotiatedAttributeOwnershipDivestiture()		theAttributes theTag
cancelNegotiatedAttributeOwnershipDivestiture()	theObject	theAttributes

表 3-12

	OWMO	Self
informAttributeOwnership()		theOwner
attributeIsNotOwned()		
attributeOwnedByRTI()		
isAttributeOwnedByFederate()	theObject	
queryAttributeOwnership()	theAttributes	

表 3-13

E、Time Management

表 3-14 中，我們可以發現在 Switch 節點下的介面函式並沒有任何共同的 attributes 可以提至上一層。

	Switch	Constrain	Regulate	Self
enableTimeConstrained()				the FederateTime Lookahead
disableTimeConstrained()				
timeConstrainedEnabled()				the Federate's current logical Time
enableTimeRegulation()				
disableTimeRegulation()				
timeRegulationEnabled()				the Federate's current logical Time

表 3-14

表 3-15 中，我們可以看到在 Advance 節點下的 attributes 分析狀況，有相同的 theTime 可以提至最上層。表 3-16 中，我們可以看到在 Query 節點下的 attributes 分析狀況，同樣的有 theTime 可以提至最上層。

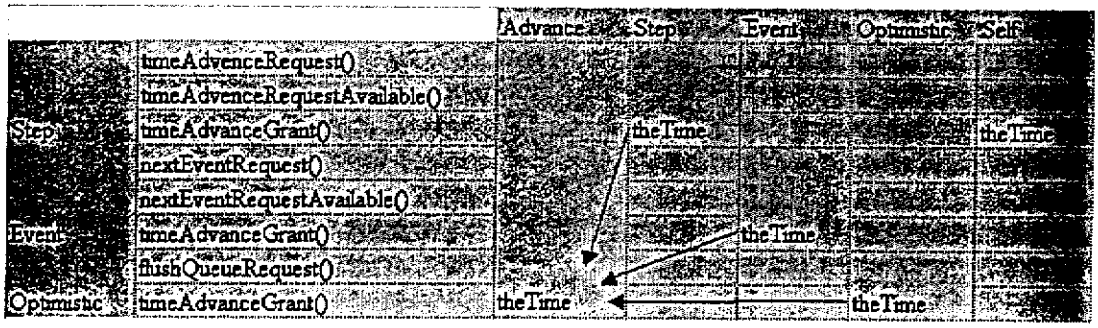


表 3-15

	Query	Self
queryFederateTime()		
queryLBTS()		
queryLookahead()		
modifyLookahead()		
queryMinNextEventTime()	theTime	

表 3-16

在表 3-17，發現這六個工具介面函式中有四個函式分別可以在 Order type 及 Synchronization delivery 的節點提出其共同的屬性。

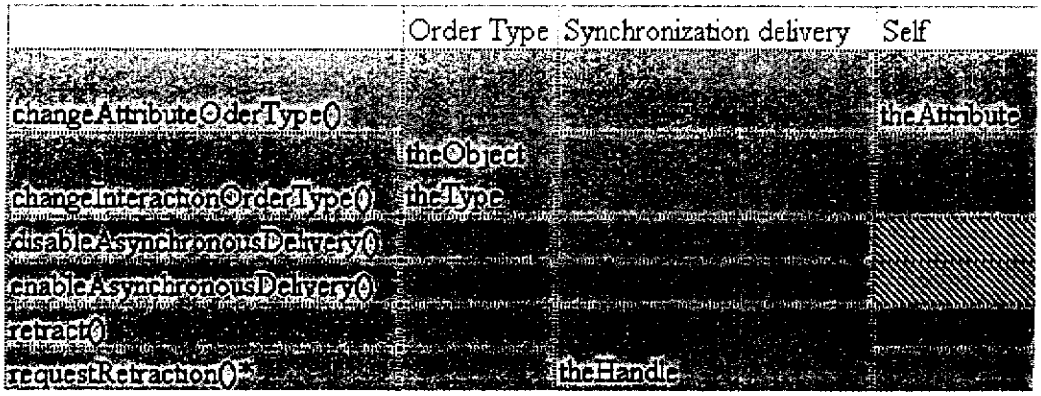


表 3-17

F、Data Distributed Management

在表 3-18 中，我們可以很清楚的看到各個介面函式經由 attributes 的過濾分析，可以提出 theRegion 至最前一層，也就是在 DDM 模組中，共同的參數為 theRegion，所以在包裝過程中，可以注意到那些參數是不須要重覆讀取的。至於在 Setting 與 Associat 節點有另外兩個共同 attribute 分屬於此節點。

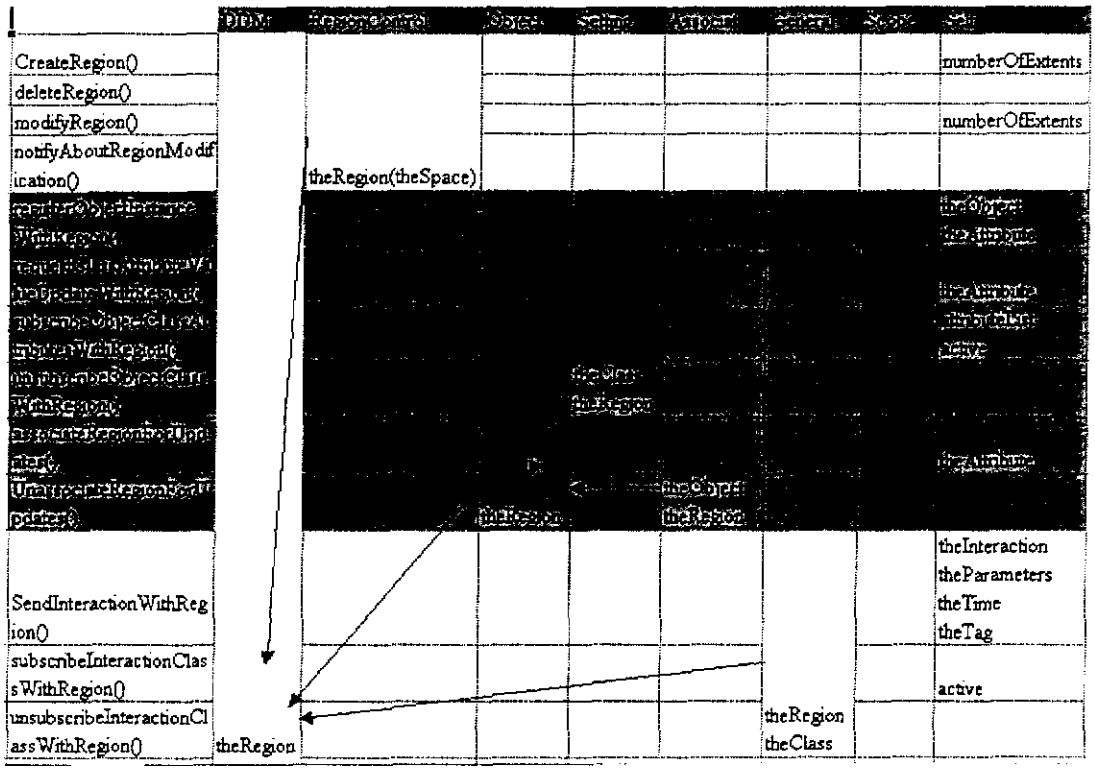

CreateRegion()
deleteRegion()
modifyRegion()
notifyAboutRegionModification()
theRegion(theSpace)
numberOfExtents
numberOfExtents
SendInteractionWithRegion()
theInteraction
theParameters
theTime
theTag
subscribeInteractionClassWithRegion()
active
unsubscribeInteractionClassWithRegion()
theRegion
theRegion
theClass

表 3-18

3.3 分析結果

利用前兩節所分析之結果，配合 RTI 1.3 Programmer Guide 所列之執行狀態，依照標準的執行步驟做出歸納分析，本計畫整合包裝出如下圖 3-29 之程式庫架構。

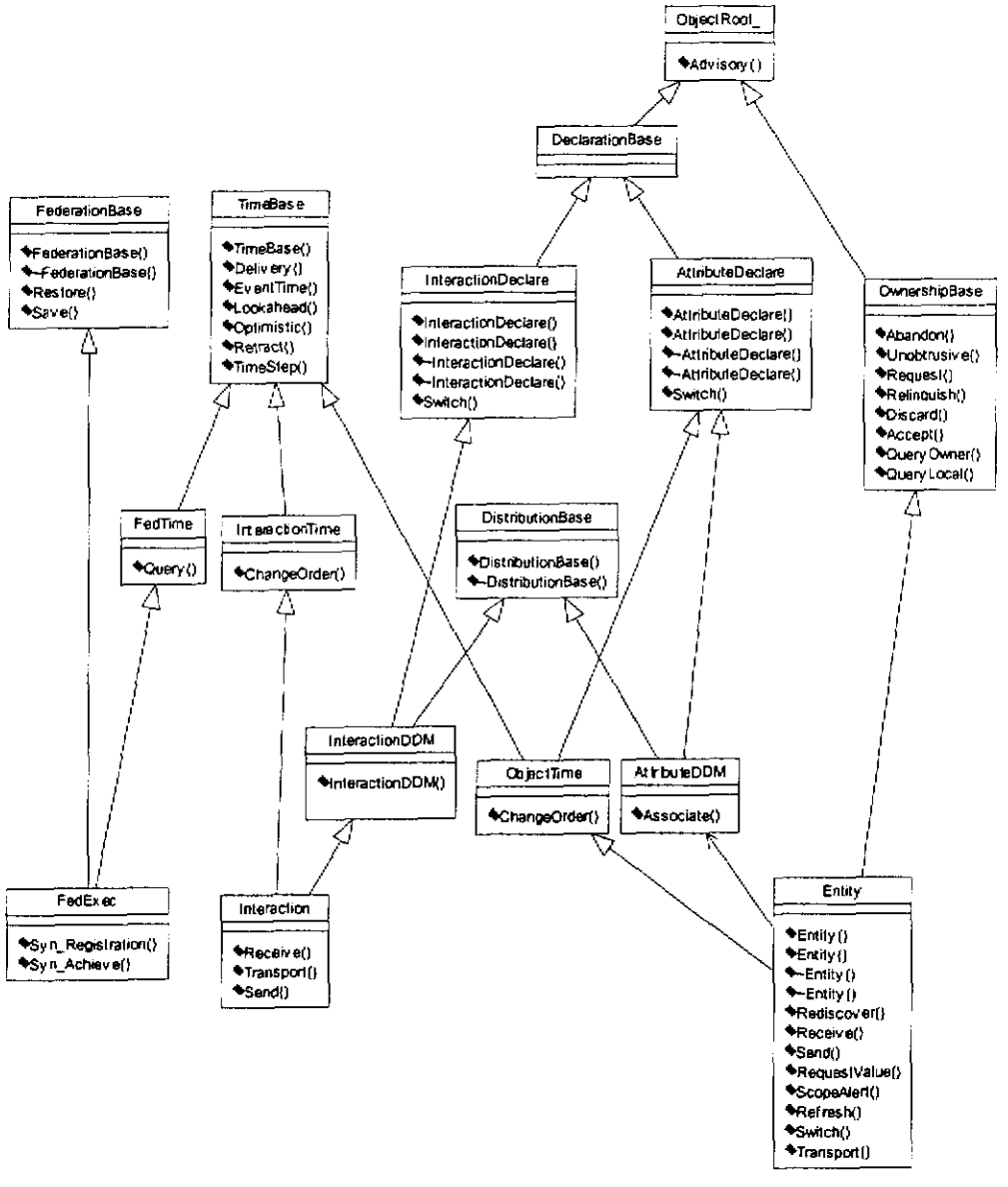


圖 3-29

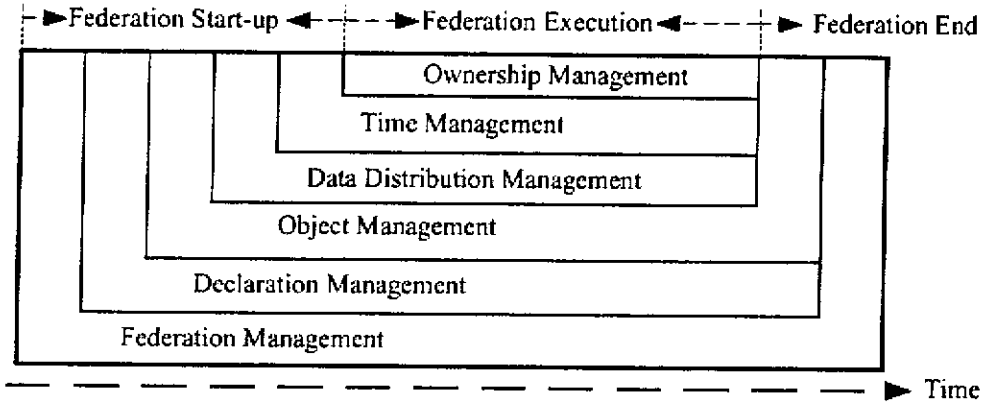
以下將分為兩個部份來解說分析包裝過程：

第一部份、首先以第 3.1 節與第 3.2 節的內容，配合 HLA 規格介面函式的功能，對六大管理模組的內容個別分析與包裝成獨立的功能函式。

第二部分、接著依照第一部份的功能函式的分析與包裝結果，再次將 3.2 節的參數分析結果提出交互比對，將個別功能函式連結成模組以建構出整體類別程式庫架構。

第一部份、功能函式分析與包裝

我們依照 RTI 1.3 Programmer Guide 所列之如圖 3-30 所示之 Federation life cycle 來逐步對 RTI 的六個管理模組的以歸納分析方式詳述如下：



The federate execution cycle

圖 3-30

A、Federation Management

我們先由第一個 Federation Management 開始，由圖 3-2 得知此 Service class 包括四個功能模組，亦即 LifeCycle, Synchronization, Save and Restore，而此四個功能模組與 HLA 其他的 services 無關，因此我們可以將其分別組合成獨立的功能函式。首先由圖 3-3 中發現與 Life Cycle 有關的四個介面函式都是由 federate 所控制，藉由它們向 RTI 告知創造 Federation 並加入新的 federate。

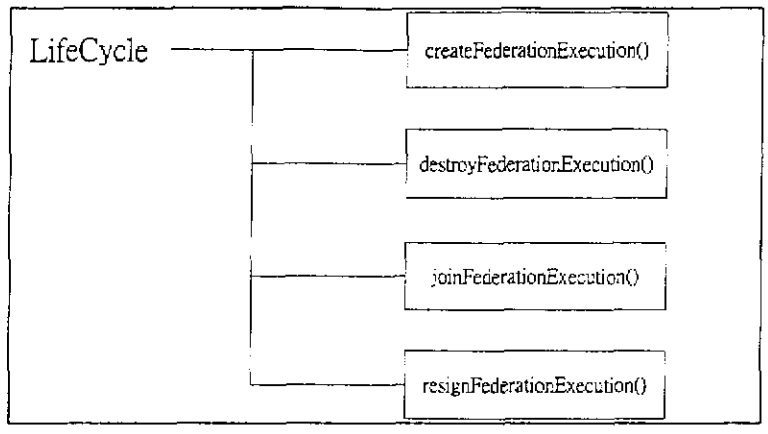
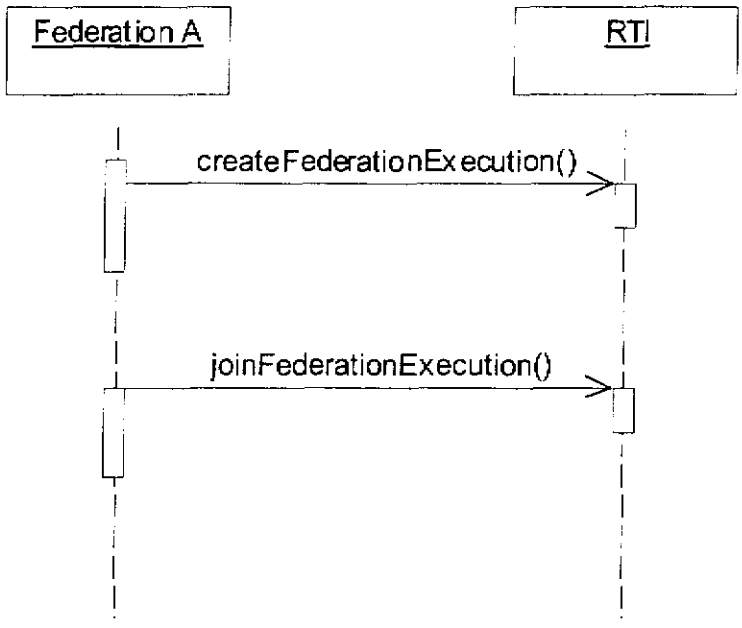


圖 3-3

因此，我們將這些步驟(或稱為介面函式)簡化為一個 class 的建構子與解構子，並且由表 3-1 中的參數可以訂出一部份 FederateBase 這個 Class 的 data member，包括：federation_name、federate_name、fed、counter 等，以及如下圖所示的建構與解構功能內容。

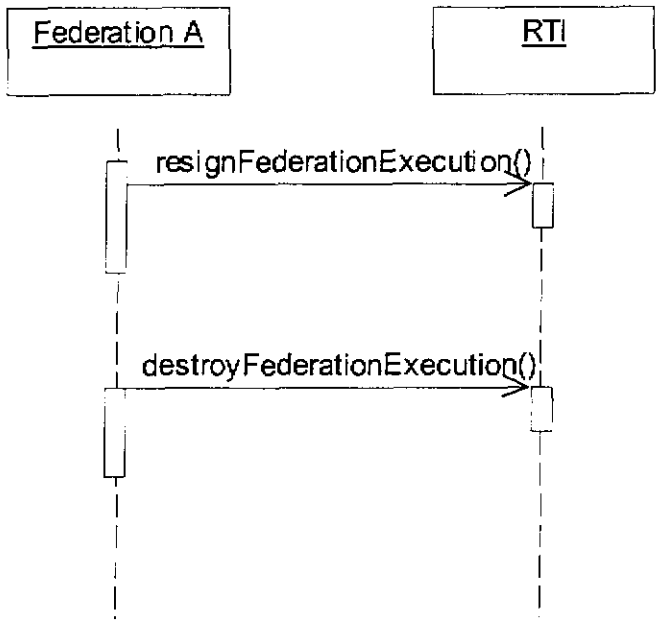
Constructor： FM class 的建構子

下圖中的兩個界面函式 Create Federateion Execution 以及 Joint Federate 為 RTI 1.3 的界面函式，我們將其包裝成 Federation Management 的 Class 的建構子，使得這個 class 在創造 instance 時，就可以將 Federation 開啟並使得這個 federate 加入。



Destructor ： FM class 的解構子

下圖中的兩個界面函式 Resign Federation Execution 與 Destroy Federation Execution 為 RTI 1.3 中的界面函式，當我們移除 Federation Management 的 class 時，可以透過解構子來自動執行這兩個界面函式的動作，以退出 Federation。



接下來，我們再看到 federation management 的其它部份，在圖 3-4 包含了處理 federate Synchronization 的介面函式。再由表 3-2 可以看到在處理

synchronization 的這些介面函式擁有共同的參數 label，代表的是 synchronization point 的名稱，因此提出當做共同的參數。

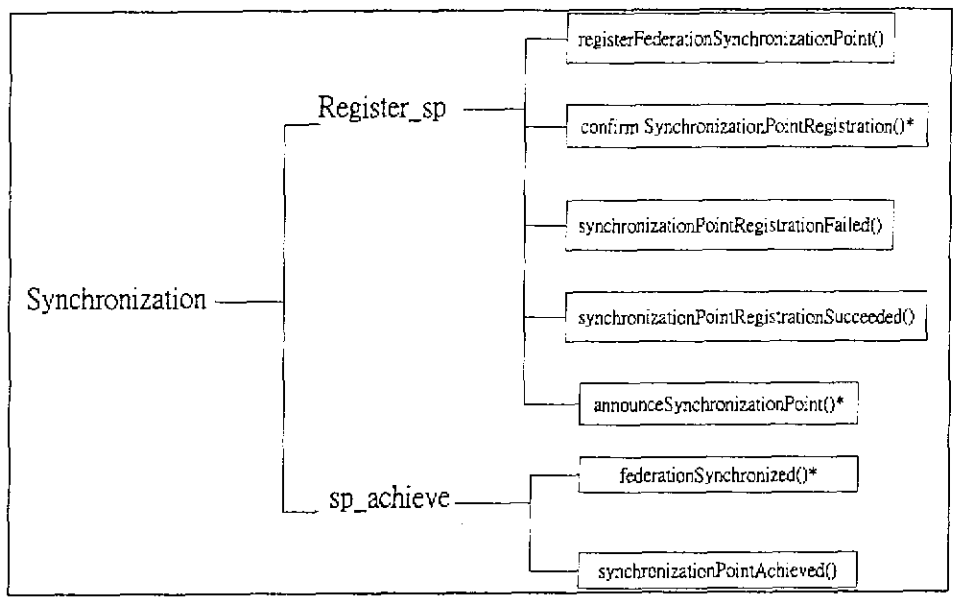
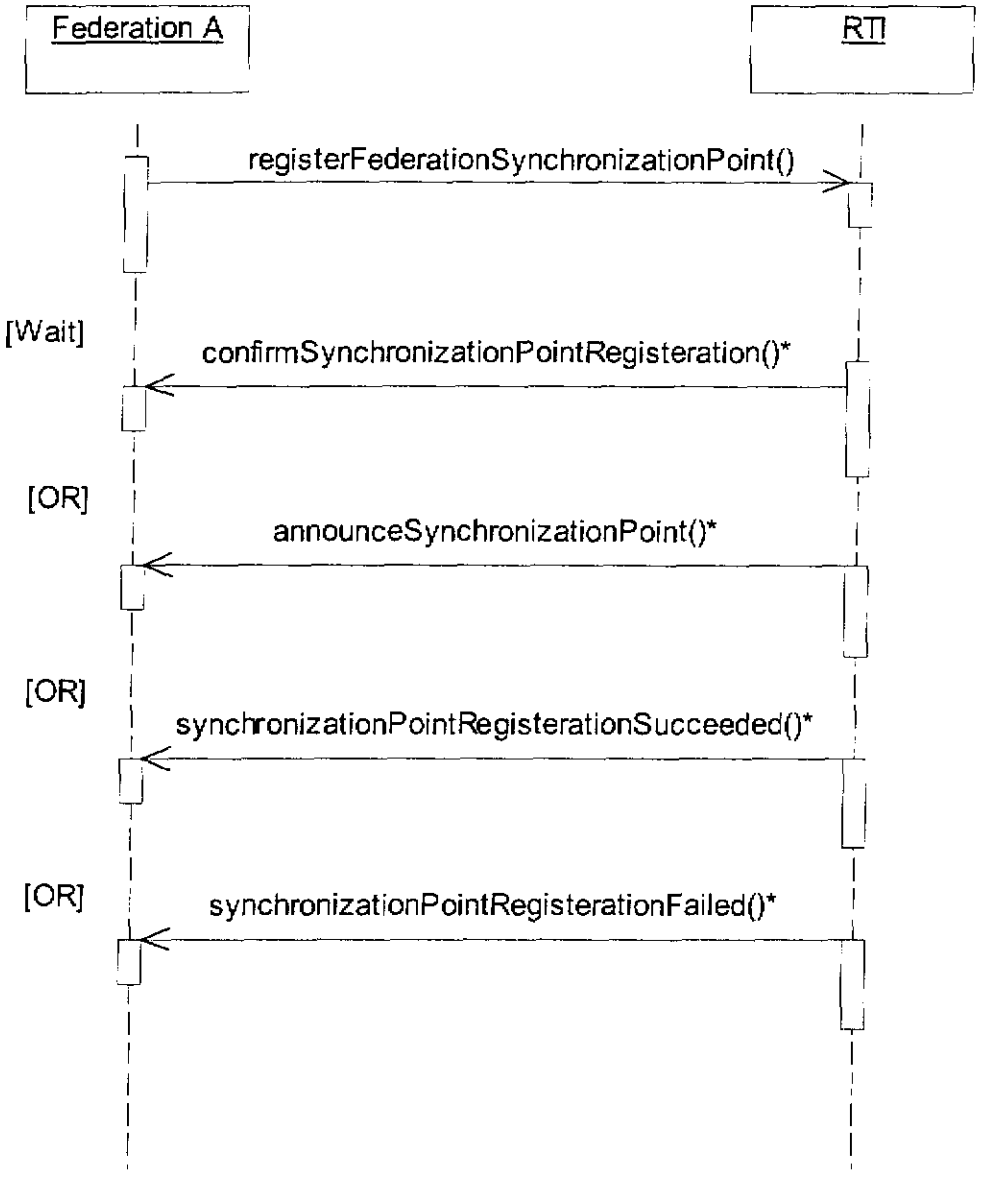


圖 3-4

由圖 3-4 我們可以將 Synchronization 的動作分為 Synchronization Point Registeration 與 Synchronization Point Achieved 兩個獨立 Function 來處理 federate synchronization 的問題，下面的執行流程圖則是說明這些 function 的功能及執行流程。

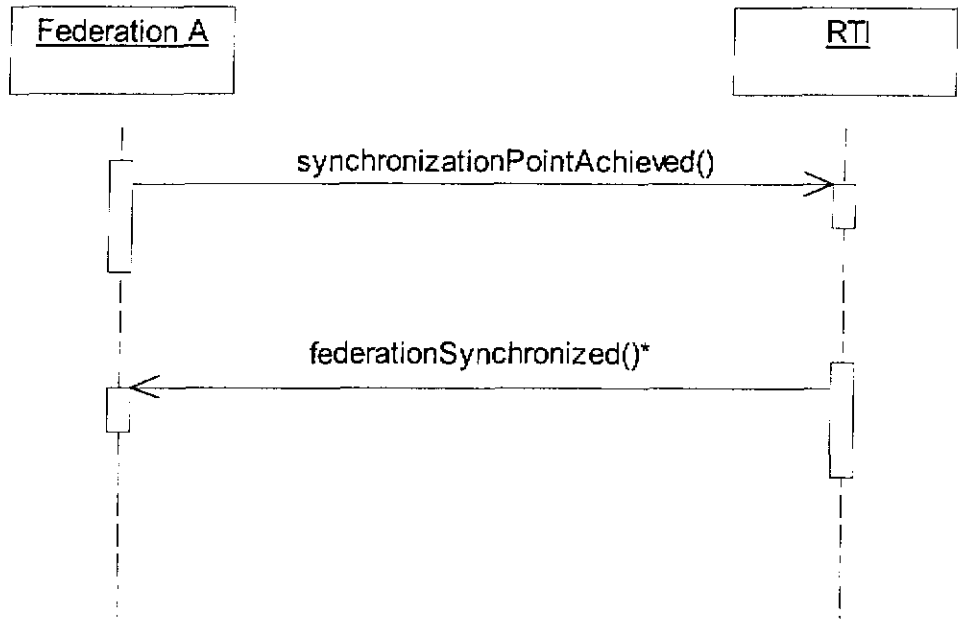
Syn Registeration：Federate 向 RTI 告知要註冊一個 Synchronization Point

這個 Function 所執行的動作如下圖所示，為 Federate 向 RTI 告知要註冊一個 Synchronization Point。接著 Confirm Synchronization Point 告知已經收到要求並開啟一個 Synchronization Point List Set，Announce Synchronization Point 為告知這個 Synchronization Point 已經被接受並加入 List Set 中，而 Synchronization Point Registration Failed 則是告知註冊錯誤或無法註冊。



Syn Achieve：Federate 告知 RTI 其已到達同步點

這個 Function 的執行時機為 federate 執行至某個設計者設定的 synchronization point 的階段，則先暫停執行，利用一個 Synchronization Point Achieved 的 Function 將這個資訊告知 RTI 並等待推進。



接著，圖 3-5 與圖 3-6 為包含了處理 federation 儲存與恢復 federation 的介面函式，由表 3-3 與表 3-4 也可以看出有共同的參數 saved label(儲存的名稱)，因此提出當成 static 參數。而這些介面函式則因為不是 federation 執行時一定要執行的功能，所以也設計成這個 class 中的 Function。

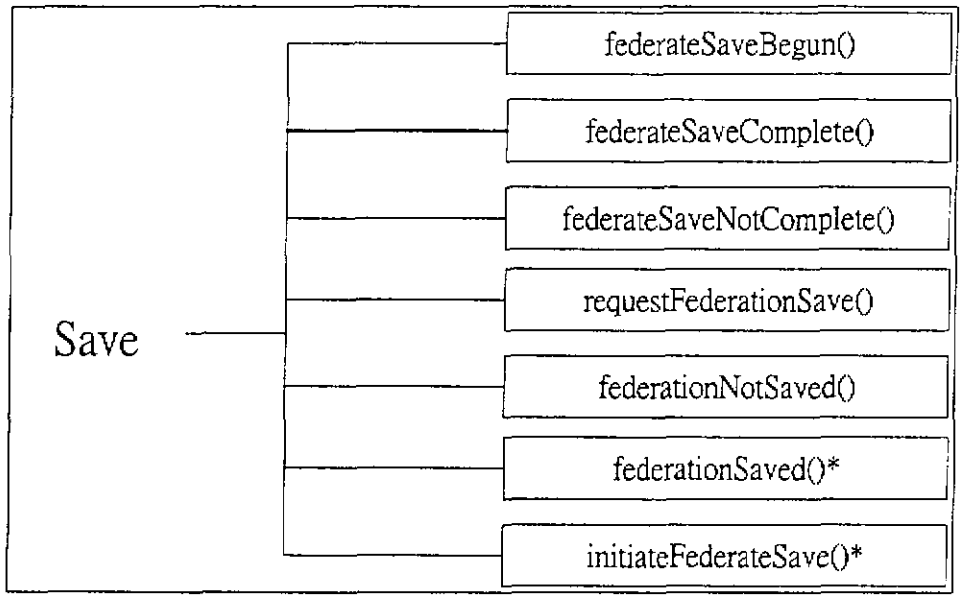


圖 3-5

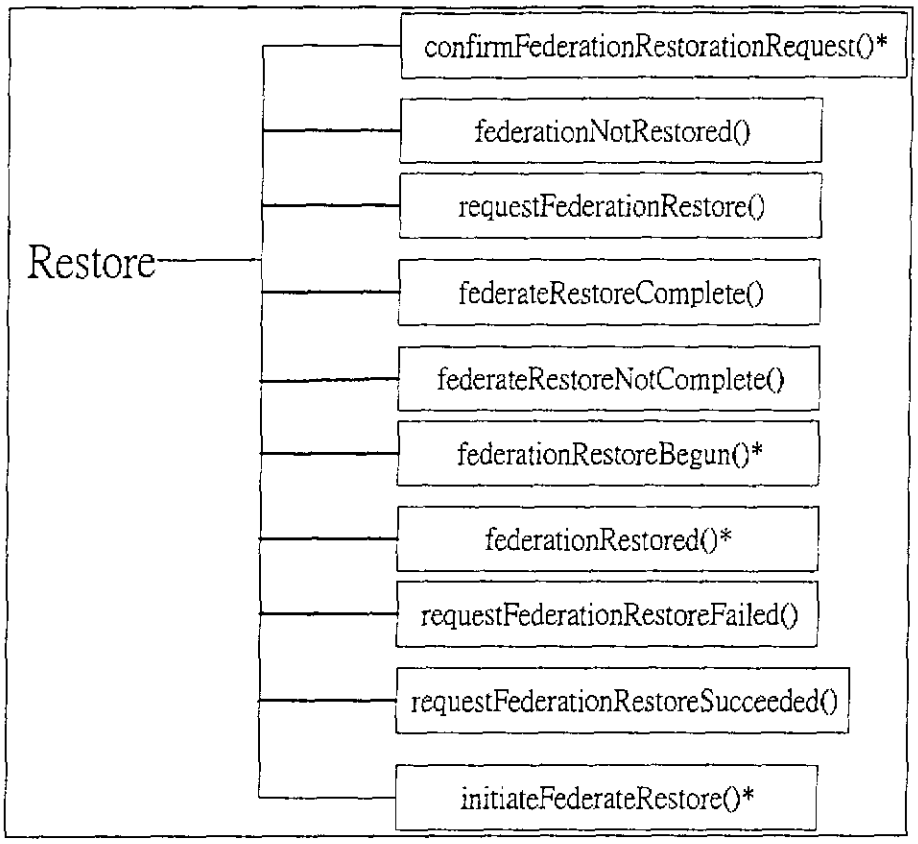
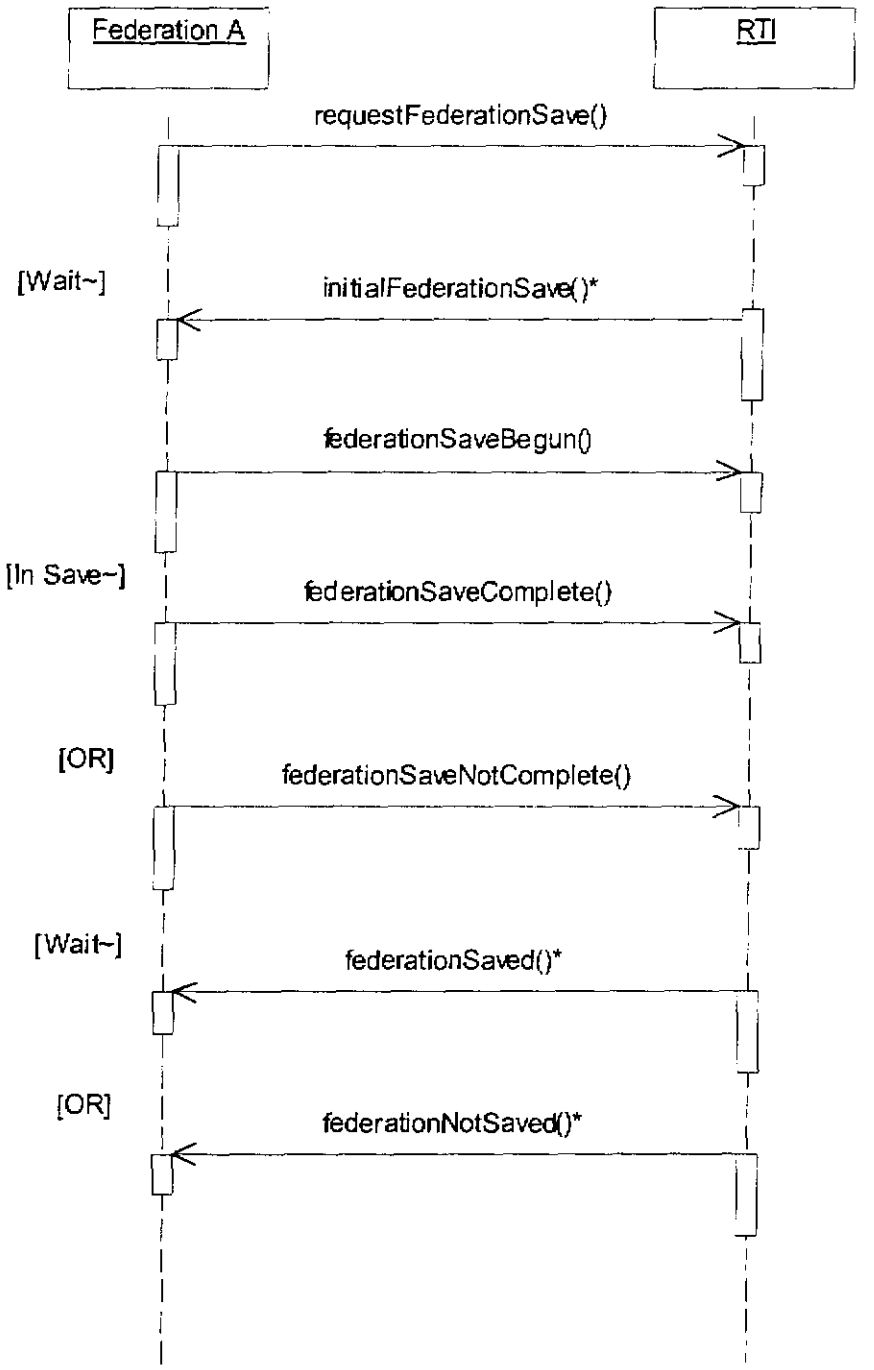


圖 3-6

因此，我們可獨立出兩個功能 functions，分別為 Save 與 Restore，如下所示。

Save：Federate 要儲存 Federation

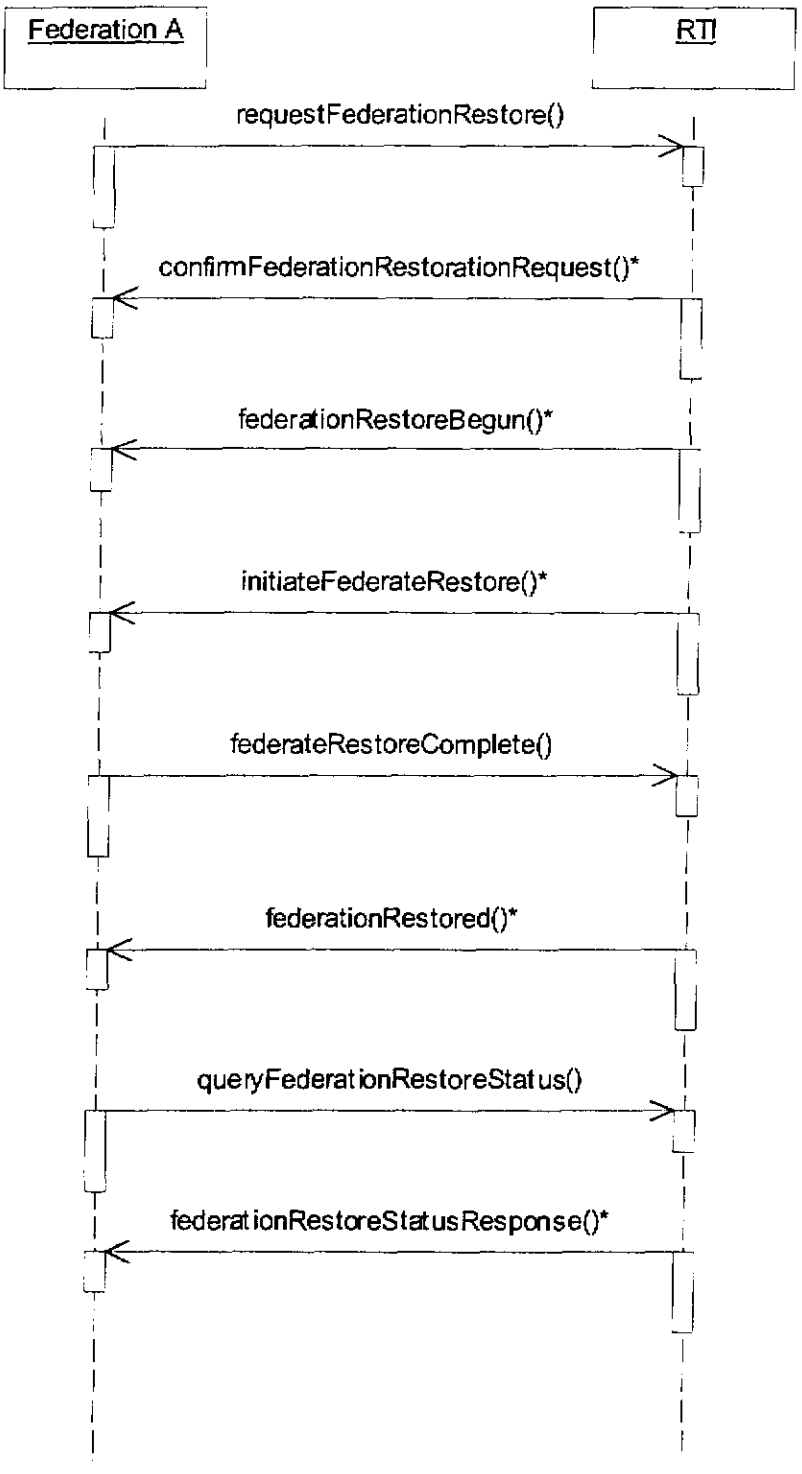
下圖說明 save 這個 Function 來執行的啟始動作，相當於 RTI 1.3 中的 Request Federation Save 界面函式。RTI 告知要求的 federate 以及其它所有 federate 成員要開始儲存，所有的動作都暫停，等儲存完畢(Federation Saved)才恢復正常的執行狀態。



Restore：Federate 要復原 Federation 至先前儲存狀態

這個 Function 的動作在執行 Federation Restore 的功能，也就是由 federate 向 RTI 要求 restore，並由 RTI 回覆要求完成。接下來 RTI 通知 federate 可以開始執行 restore， 而 Federate 在接收到訊息後開始做 restore 的動作，並告知 RTI 所有動作暫停。最後，於 Federate 做完 Restore 的動作後，向 RTI 告知是否成

功。當 RTI 收到所有 Federate 的通知已經完成 restore 動作，則發送通知告訴所有 federate 可以繼續恢復執行的動作。



B、Declaration Management

分析完 Federation Management 後，我們接下來對圖 3-30 中 federation execution cycle 的第二個順序的 Declaration Management 分析，經由圖 3-7 的分析可以知道這個 management 分為兩大類，一是 Object(也就是以 Object 為對象做宣告)，另一為 Interaction(也就是以 Interaction 為對象做宣告)。

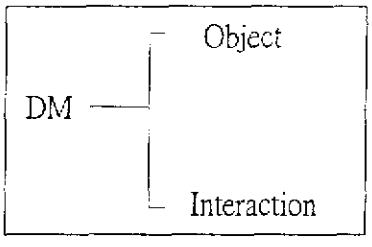


圖 3-7

另外，透過表 3-5 與表 3-6 可以看得出擁有共同的參數，分別是 Object 的 theClass 以及 Interaction 的 theClass。因此，我們以 DeclareBase 為基本的 Class 將前面提及的共同參數提出。

(子節點)		Object	publish	subscribe	register	Self
publish	publishObjectClass()		theClass			attribute
	unpublishObjectClass()					
subscribe	subscribeObjectClassAttributes()					attribute active
	unsubscribeObjectClass()			theClass		
register	startRegistrationForObjectClass()	theClass			theClass	
	stopRegistrationForObjectClass()					

表 3-5

(子節點)		Interaction	Publish	Subscribe	turn	Self
publish	publishInteraction()		theInteraction			
	unpublishInteractionClass()					
subscribe	subscribeInteractionClass()			theClass		active
	unsubscribeInteractionClass()					
switch	turnInteractionsOn()				theHandle	
	turnInteractionsOff()					

表 3-6

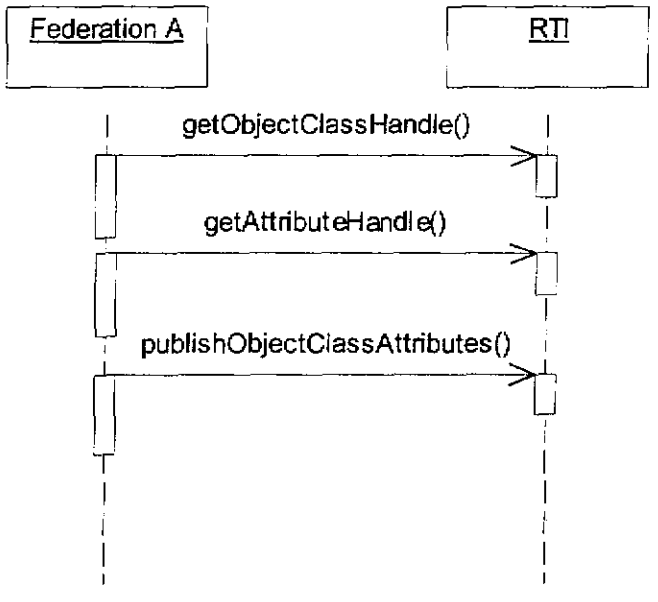
然後衍生出兩個 subclasses，分別為與 Object 有關的 class 以及與 Interaction 有關的 class，分別包含處理 Object 宣告以及 Interaction 宣告等的 Member Function。

在與 Object 有關的這個 class 當中，除了需要建構子與解構子的 Function 來宣告物件型態，依據圖 3-8 得知，建構子與解構子的設計須區分為 publicher 與 subscriber 兩類，以及包括一個 Switch function 來只是何時可以開始註冊物件，下面分別介紹 Object class 的建構子與解構子的功能與執行流程。

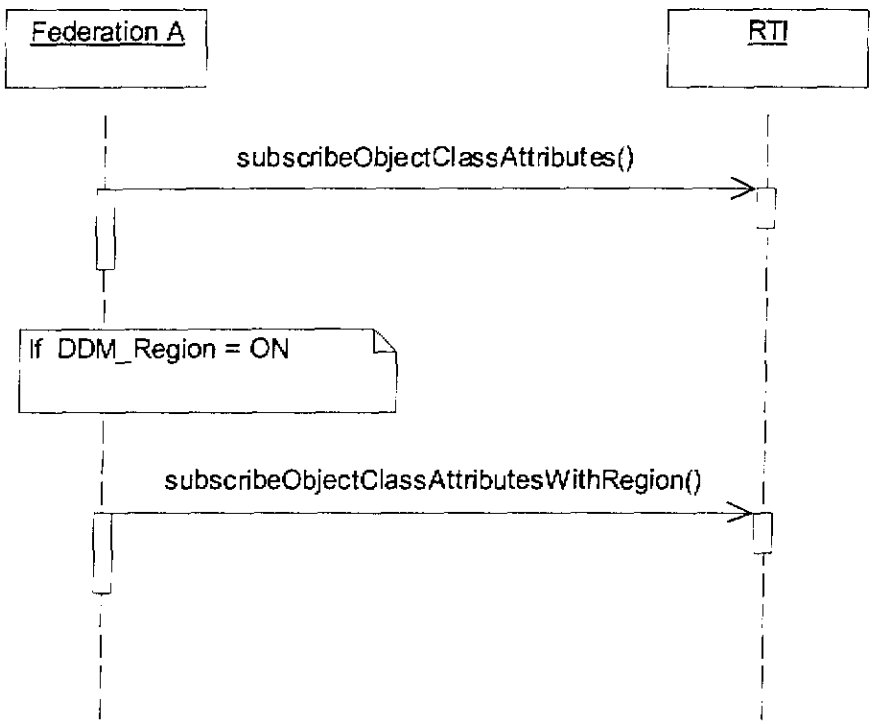
Constructor：與 Object 相關的建構子

這個 Function 透過 federate 的宣告型態來宣告所要產生的資料型態，或是所想要接受之資料型態。

Publish



Subscribe

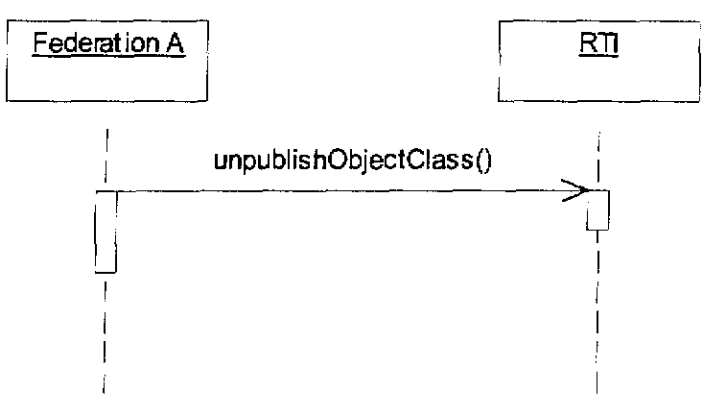


上述與 subscribe 有關的建構子中包含 Data Distribution Management 的 service call 功能，是因為 RTI 的功能模組允許 federate 根據物件資料型態的影響範圍來接收，因此將 Data Distribution Management 的 subscribe 介面函式列於此處。

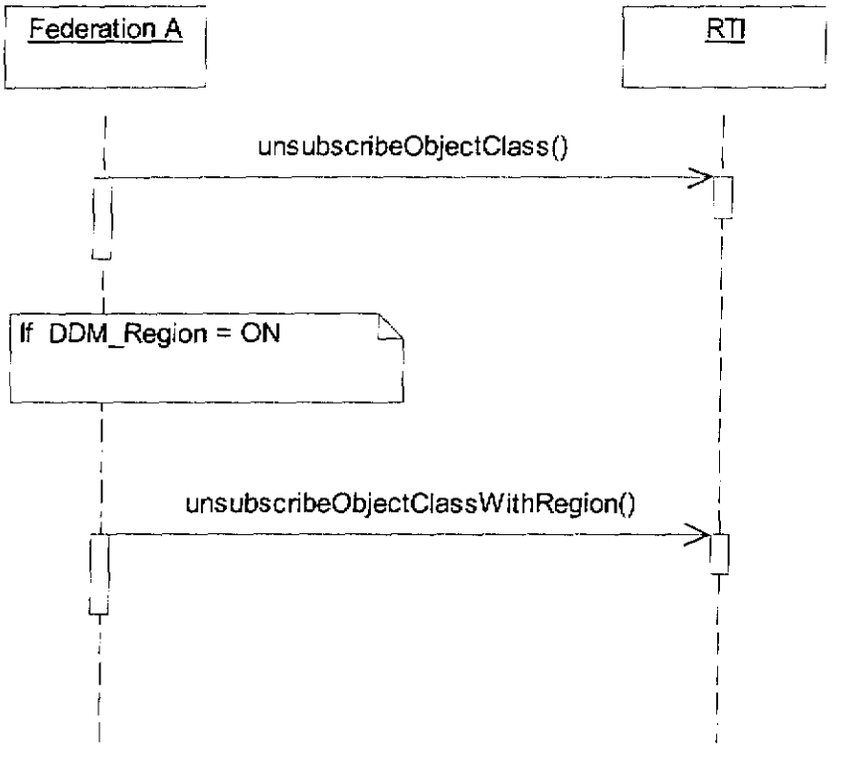
Destructor：與 Object 相關的解構子

這個解構子的動作為取消 Object 的宣告，同樣分為 publisher 與 subscriber 兩類來自動宣告解除 publish 或是 subscribe 某個 Object 的資料型態。

Publisher



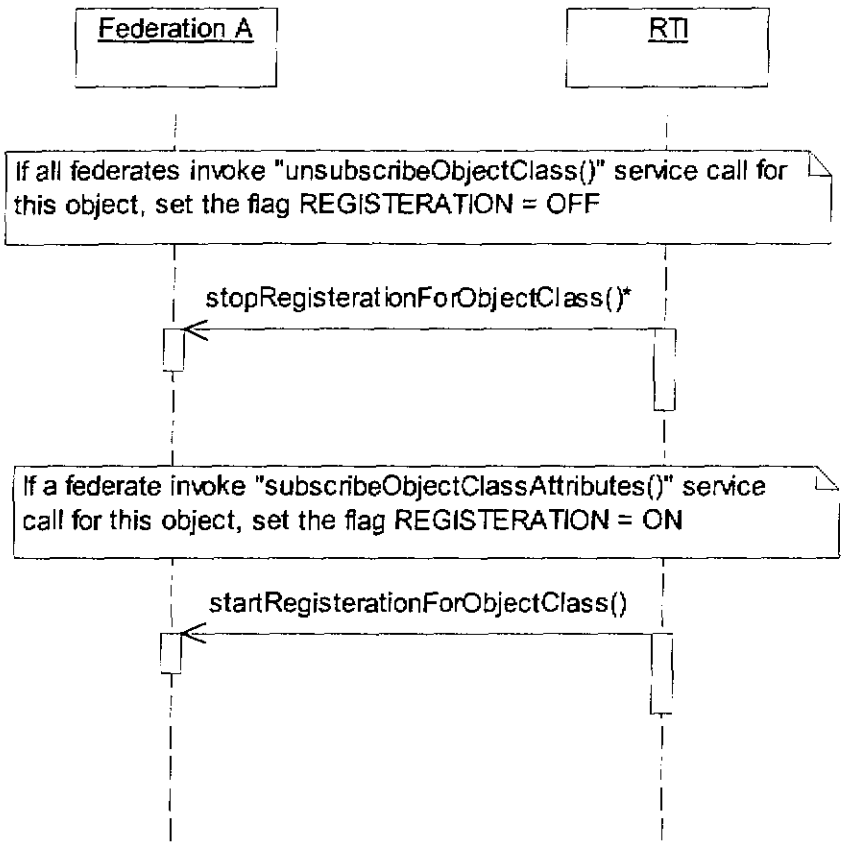
Subscriber



同樣的，與 subscribe 有關的解構子中也須包含 Data Distribution Management 的 service call 功能以配合建構子內相同功能的成對出現。

Switch：與 Object 相關的 Switch

此功能模組是用來提昇系統效能，藉由 RTI 的主動通知，federate 可以得知何時可以開始產生物件模型？以及何時可以可以停止產生物件模型？

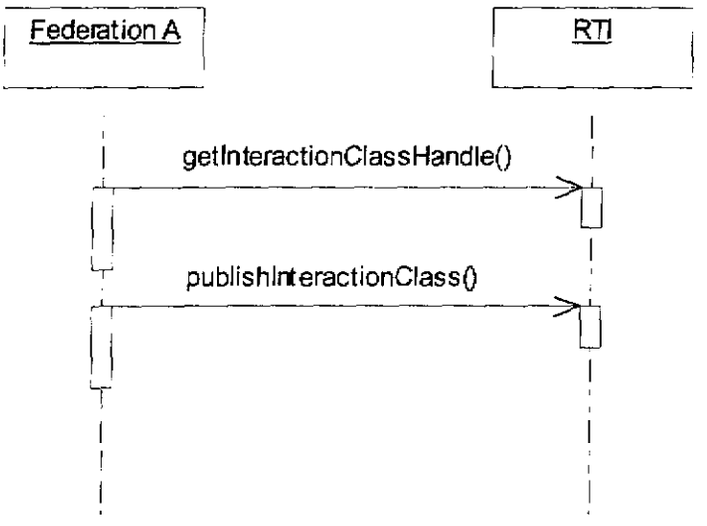


同理可知，與 Interaction 有關的這個 class 除了需要建構子與解構子的 Function 來宣告互動事件型態，依據圖 3-9 得知，建構子與解構子的設計須區分為 publicher 與 subscriber 兩類，以及包括一個 Switch function 來只是何時可以開始註冊物件，下面分別介紹 Interaction class 的建構子與解構子的功能與執行流程。

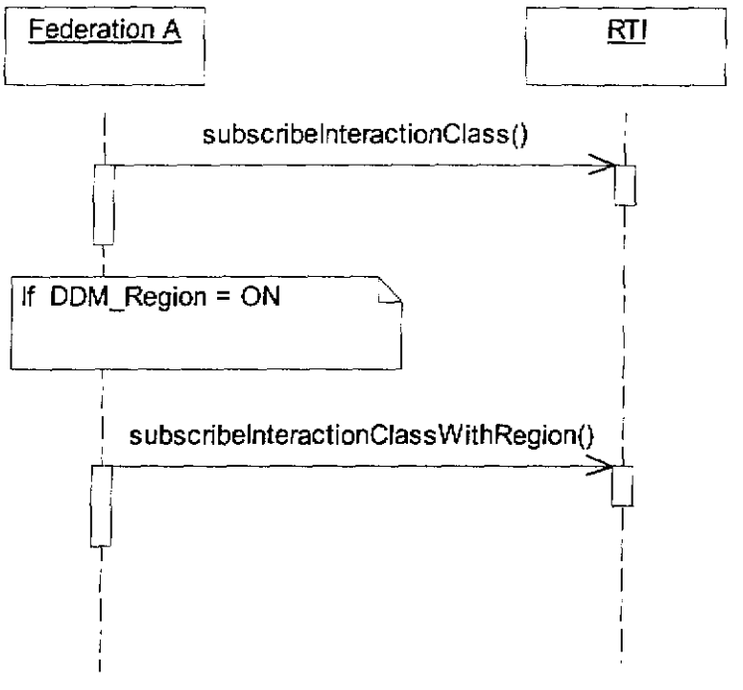
Constructor：與 Interaction 相關的建構子

這個 Function 透過 federate 的宣告型態來宣告所要產生的互動事件型態，或是所想要接受之互動事件型態。

Publish



Subscribe

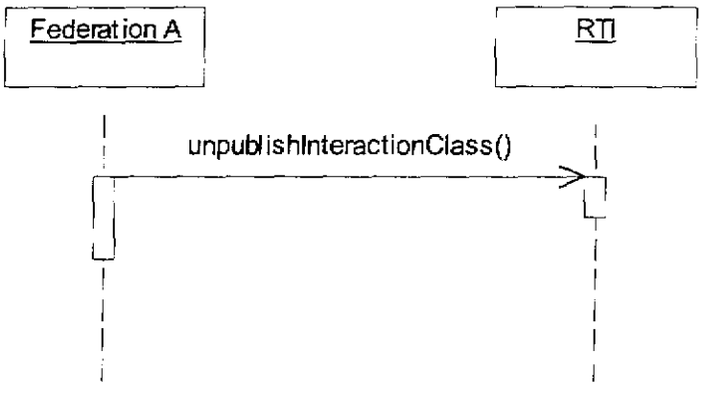


上述與 subscribe 有關的建構子中包含 Data Distribution Management 的 service call 功能，是因為 RTI 的功能模組允許 federate 根據互動事件型態的影響範圍來接收，因此將 Data Distribution Management 的 subscribe 介面函式列於此處。

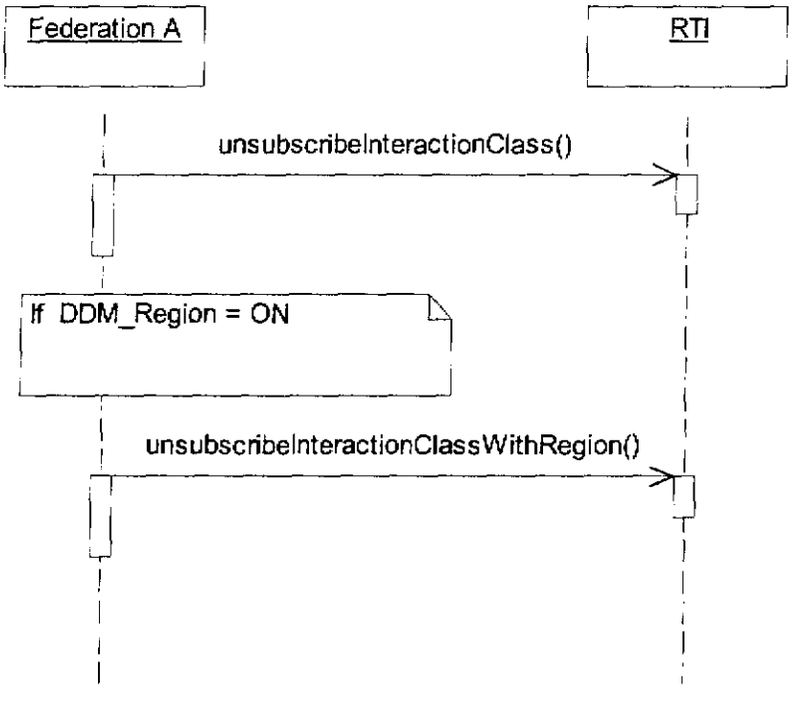
Destructor：與 Interaction 相關的解構子

這個解構子的動作為取消 Object 的宣告，同樣分為 publisher 與 subscriber 兩類來自動宣告解除 publish 或是 subscribe 某個 Object 的資料型態。

Publisher



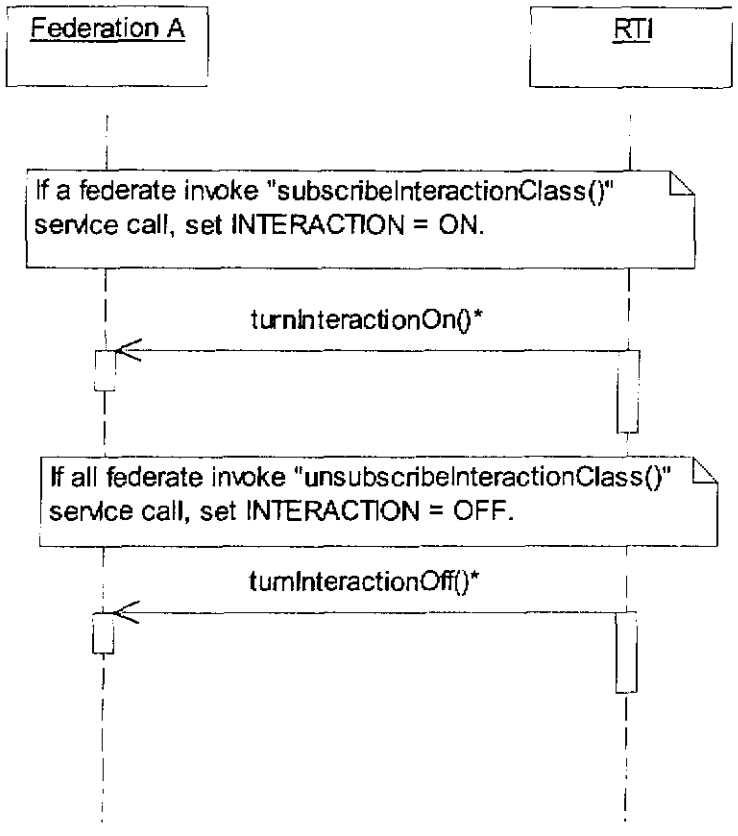
Subscribe



同樣的，與 subscribe 有關的解構子中也須包含 Data Distribution Management 的 service call 功能以配合建構子內相同功能的成對出現。

Switch：與 Interaction 相關的 Switch

此功能模組是用來提昇系統效能，藉由 RTI 的主動通知，federate 可以得知何時可以開始產生互動事件？以及何時可以可以停止產生互動事件？



C、Object Management

接著我們分析 Object Management，由圖 3-10 可以得知分為物件實體(Instance)部份、執行 attribute 資料傳遞與宣告部份以及 Interaction 部份。

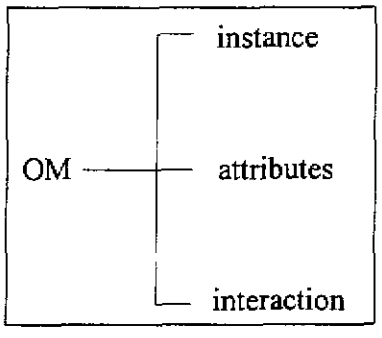


圖 3-10

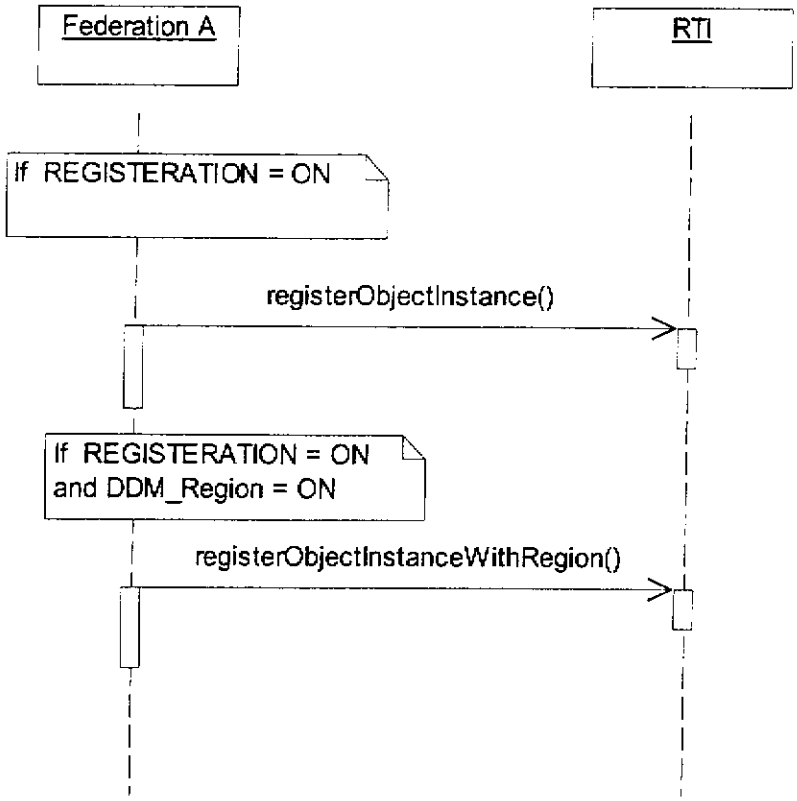
再經由表 3-7、表 3-8 與表 3-9 的分析可以得知此服務類別(Service class)分別屬於兩大類，一是 Object，另一是 Interaction。所以我們將這個部份分為兩個 SubClasses。但是，我們再由 federation 的執行架構來思考，會發現 Object 與 Interaction 是在模擬過程中最基本的單位，但是它們卻也與其他的 Management 息息相關，必須依賴其它模組的完成才可以執行，例如一個 Object Instance 必須先經過 Object 被宣告成 Publish 或 Subscribe 後才可以執行創造 Instance 的動作。因此，我們考慮將這兩個 subclass 當成最底層的 class。而 Object 這個 class 的名稱因為與 C++中的定義衝突，因此改為 Entity。

也因為上述的一些原因，包含在這兩個底層 class 中的成員將會擁有繼承而來的建構子與解構子，提供這個基本單位元件(entity)的初始化設定；以及在執行階段所需要的基本 function 來執行相關的功能。此外，由於 Object Management 的使用必須配合 Declaration Management，因此本服務類別的分析也必須分為 publish 與 subscribe 兩類來探討。以下分為 Object 與 Interaction 來介紹所分析的功能模組。

Constructor：與 Entity 相關的建構子

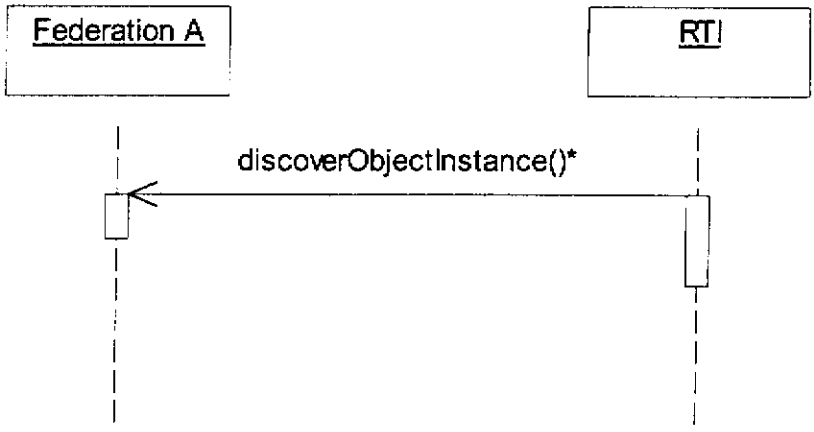
這個建構子的功能在於註冊一個 Instance，並通知 RTI，而 RTI 會告知層宣告的 Federate 發現新的 Instance。RTI 也會告知原 Federate 開啟 Update 機制。

Publish



上述與 publish 有關的建構子中包含兩個 HLA 規格所提供用來提昇執行效率的功能，而我們可將此功能以 switches 方式來實現，一個是 REGISTERATION，另一個是 DDM_Region。其中的 REGISTERATION 是要讓 RTI 可以主動通知 federate 何時可以開始產生物件？而 DDM_Region 是與 Data Distribution Management 有關的限定所產生之物件的有效影響範圍。

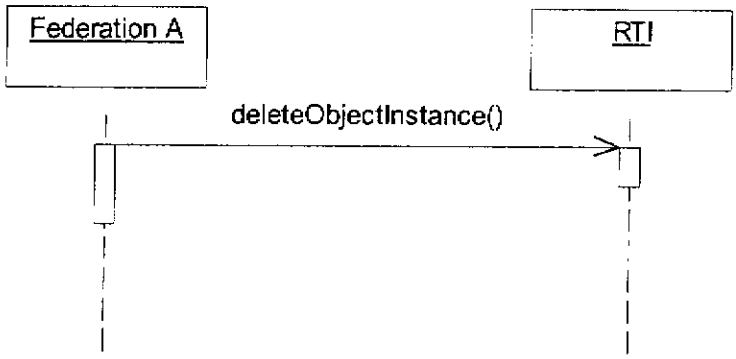
Subscribe



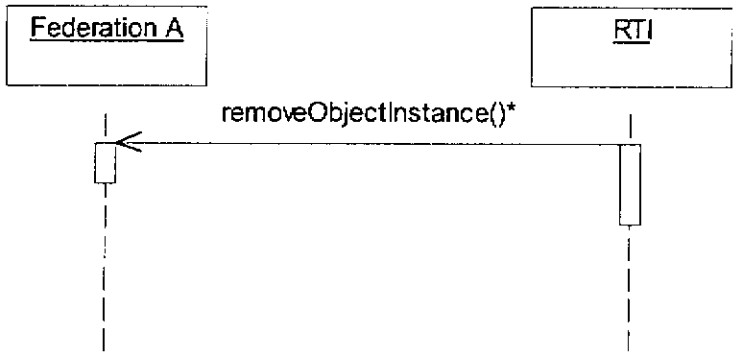
Destructor：與 Entity 相關的解構子

這個解構子的功能在在 publish 端是於 Delete 一個 Instance，並通知 RTI，而 RTI 會告知層宣告的 Federate 移除這個的 Instance。而在 subscribe 端是由 RTI 也知會 federate 其所 subscribe 的物件以不存在了。

Publisher

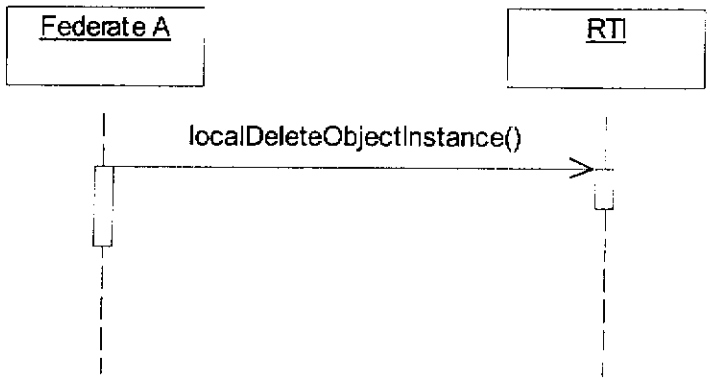


Subscriber



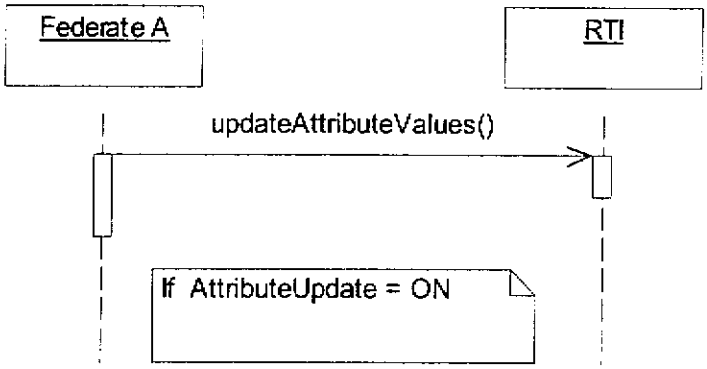
Rediscover：與 Entity 相關的重新宣告功能

此外，HLA 規格也提供一個功能讓 Federate 再次與其他 Federate 確認其所擁有物件的功能，此功能稱為 localDeleteObjectInstance ()，因此我們以一個較易記得的名詞來包裝如下。



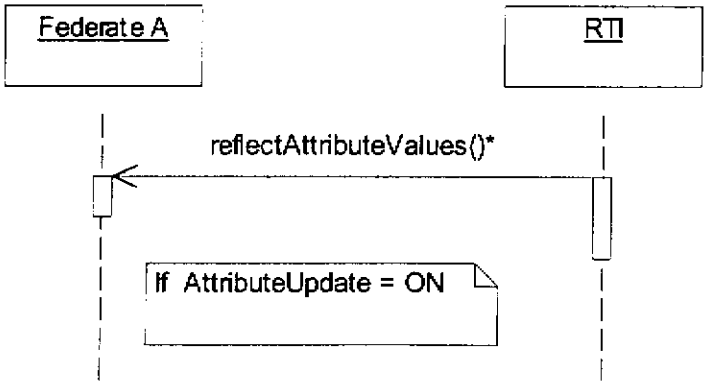
Send：與 Entity 相關的主動傳送(Send)物件訊息

這個 Function 的功能是 Federate 主動送出物件資料更新給 RTI。



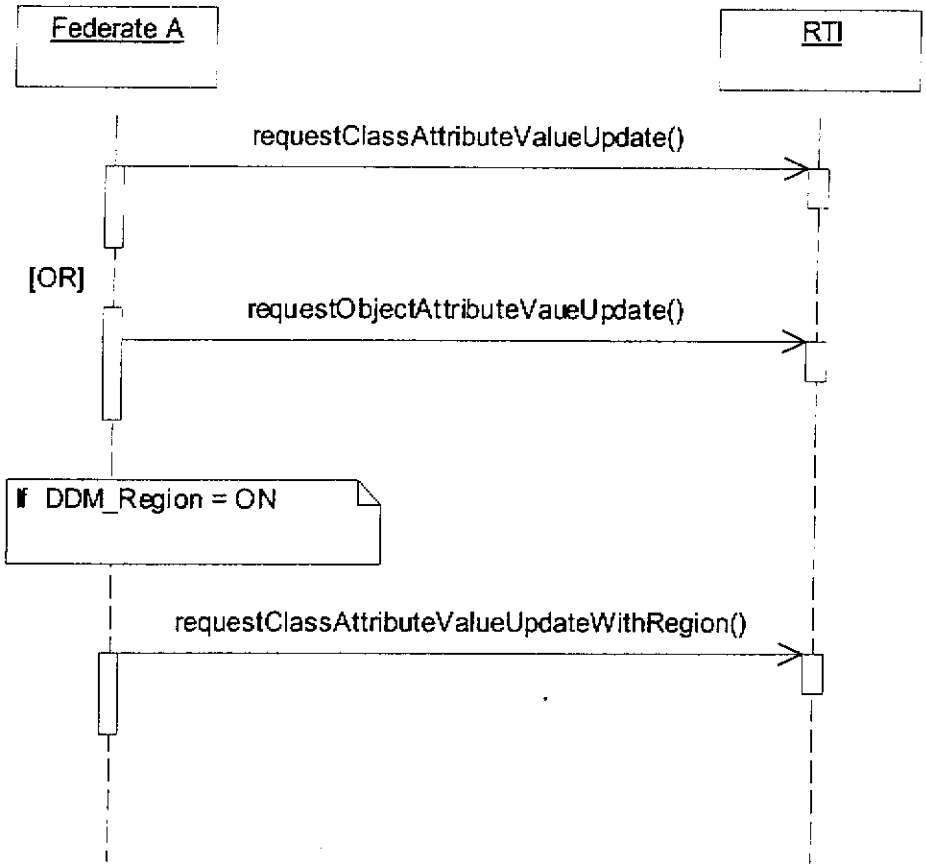
Receive：與 Entity 相關的主動接收(Receive)物件訊息

這個 Function 是在處理 RTI 所傳來之物件更新資料，使得 federate 可以更新所 subscribe 的物件。



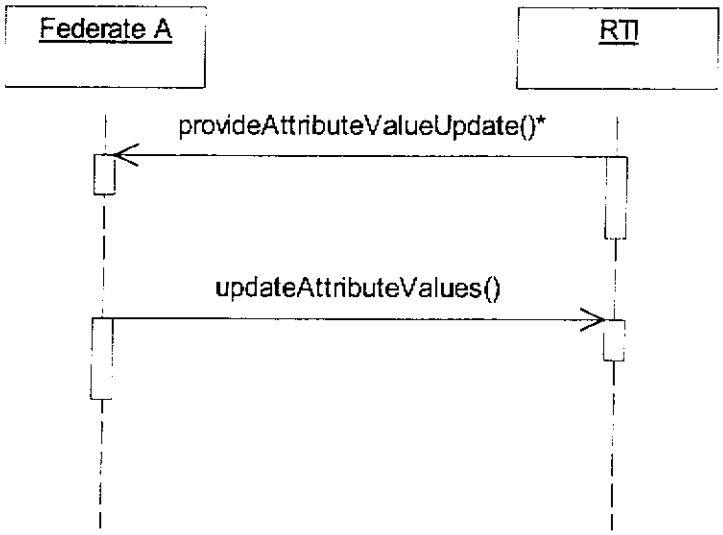
RequestValue：與 Entity 相關的被動要求(Request)物件訊息

這個 Function 是在 Federation 執行中，由 Federate 向 RTI 提出需要更新所 subscribe 物件的資料。而 Federate 提出資料更新要求有三種型態，一個是要求某個類別的所有物件資料更新，另一個是某特定物件的資料更新，最後一個是與 Data Distribution Management 有關的依資料影響範圍的資料更新。



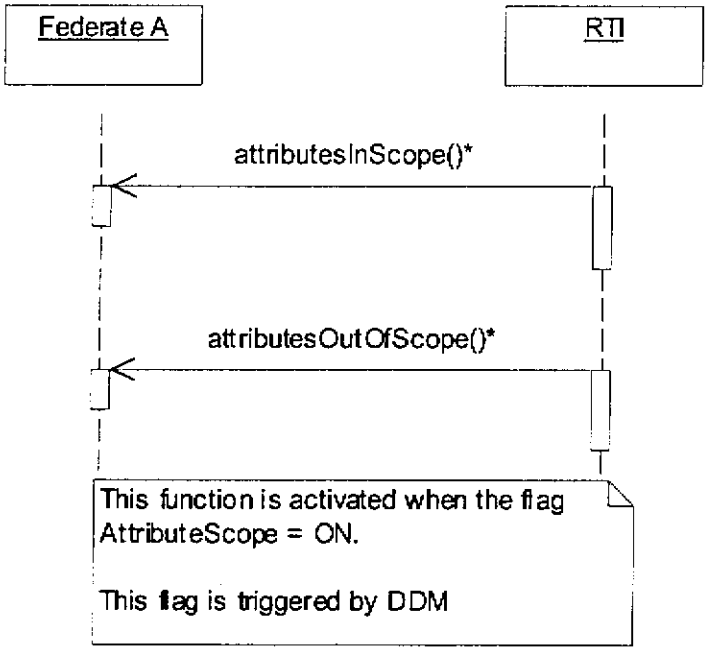
Refresh：與 Entity 相關的被動提供物件訊息

接續上面一個 Function 的動作，RTI 收到要求後，會向擁有此物件的 Federate 轉送出此要求，希望能得到新的資訊。由於 Federate 可能已經送過此物件的最新資料，此次的要求等於是要他重送資料，因此我們稱為 Refresh，而 Fedeate 收到由 RTI 通知的資料更新要求，則會將新的資訊送出。



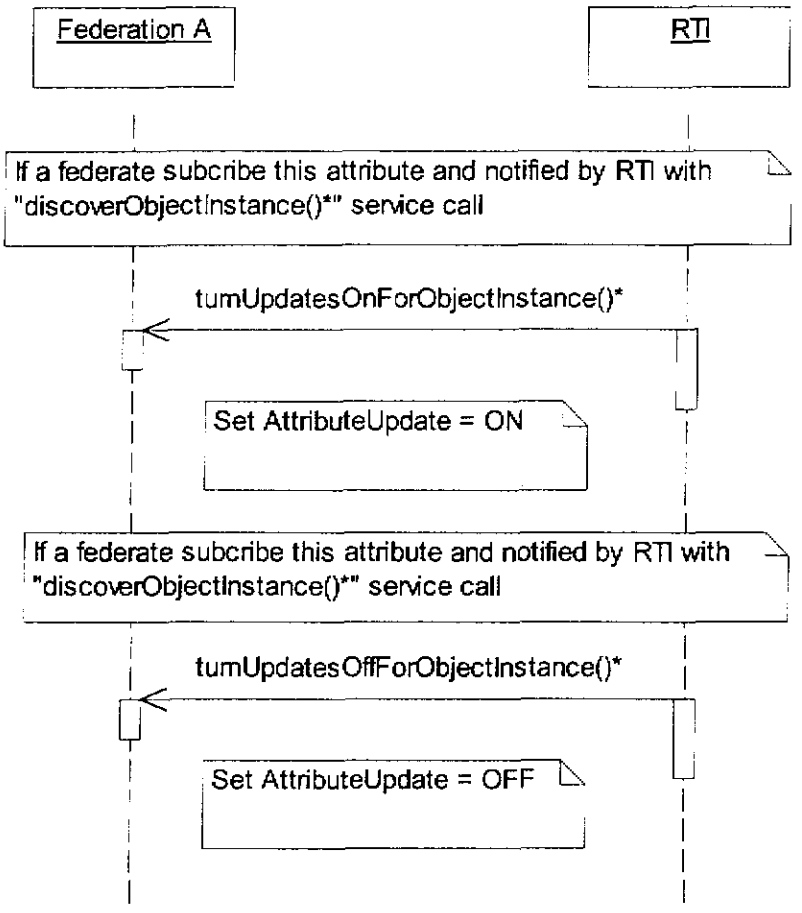
ScopeAlert：與 Entity 相關的 DDM 功能

此功能是實現 HLA 規格中允許 RTI 主動知會 Federate 其所 subscribe 的物件是否出現在資料影響範圍？此功能必須與 Data Distribution Management 配合使用，由 DDM 的 service call 定義資料影響範圍。



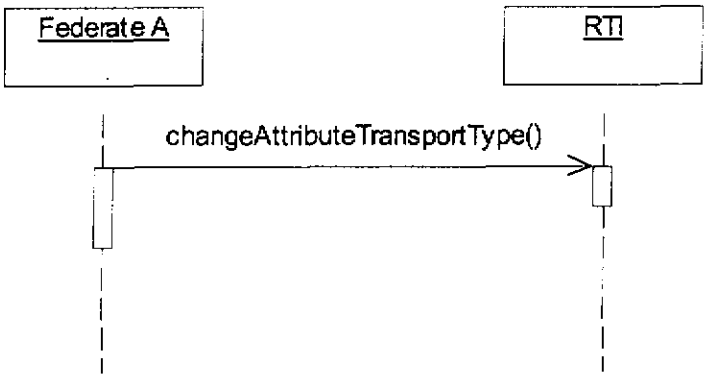
Switch：與 Entity 相關的啟動資料傳送功能

此功能也是 HLA 規格所定義之提昇效能的功能，由 RTI 主動通知 Federate 其所擁有的物件是否開始傳送物件的資料。



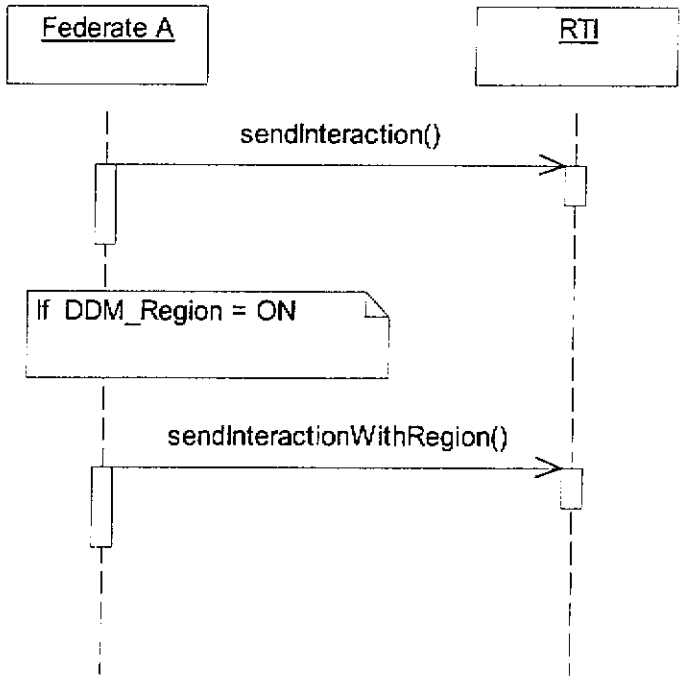
Transport：與 Entity 相關的改變資料傳送方式的功能

HLA 規格允許 Federate 在執行過程動態改變資料傳送方式，因此我們也將此功能包裝成易懂的名稱。



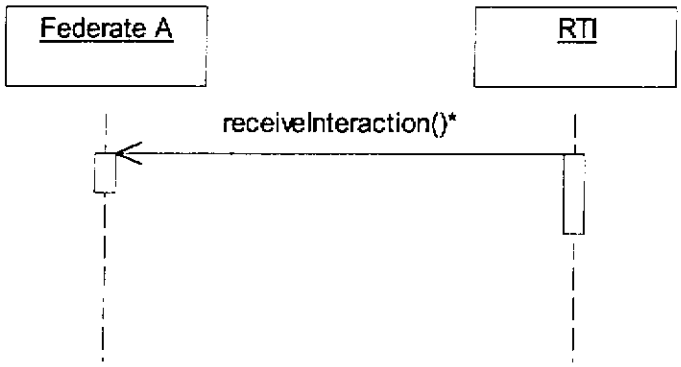
Send：與 Interaction 相關的事件訊息傳送(Send)功能

此功能讓 Federate 主動送出所產生的事件訊息，而與 Object 相關，此事件訊息傳送也可以配合 Data Distribution Management 設定成與資料影響範圍有關，來決定誰會收到訊息。



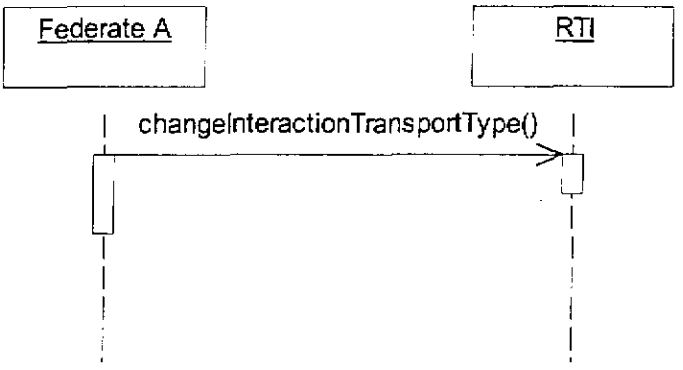
Receive：與 Interaction 相關的事件訊息接收(Receive)功能

這個 Function 是在處理 RTI 所傳來之事件訊息資料，使得 federate 可以得知所 subscribe 的事件發生情形。



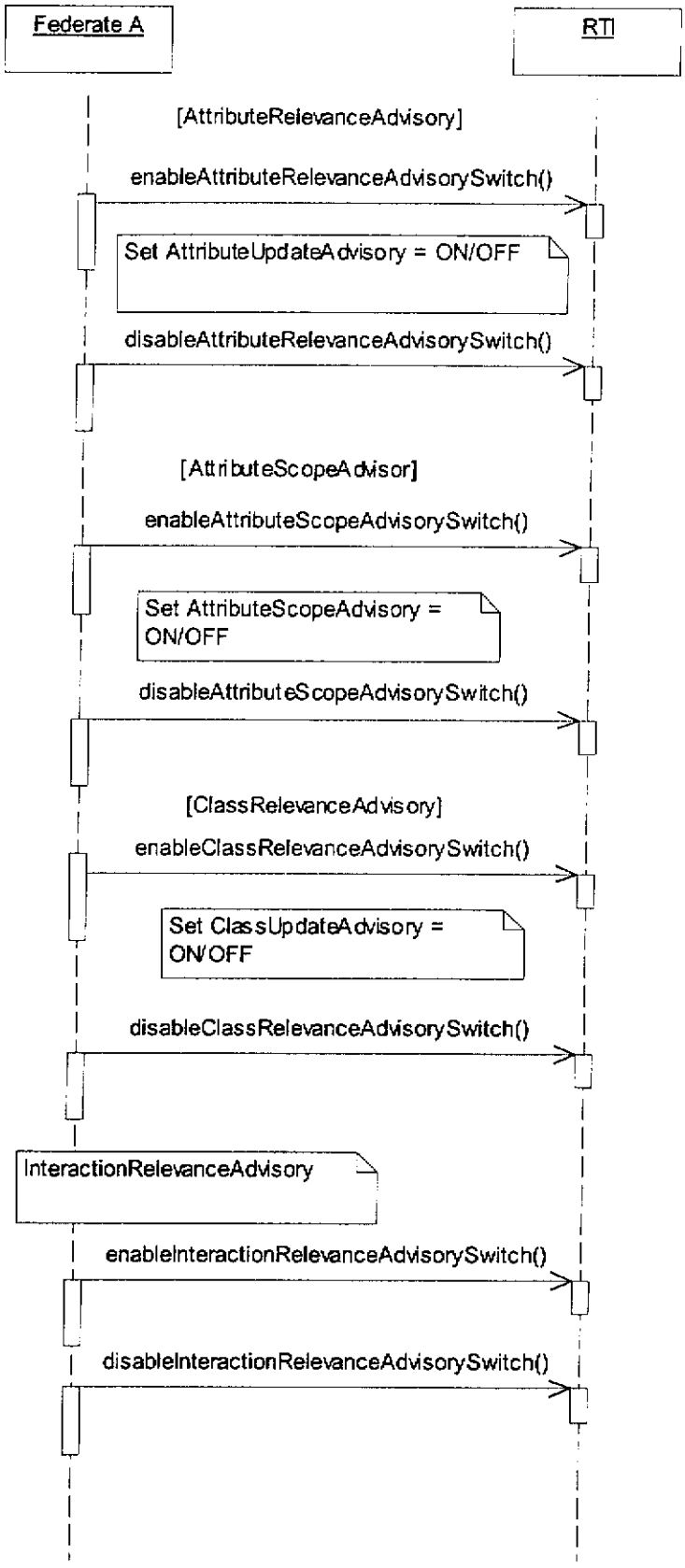
Transport：與 Interaction 相關的改變事件資料傳送方式的功能

HLA 規格允許 Federate 在執行過程動態改變資料傳送方式，因此我們也將此功能包裝成易懂的名稱。



Advisory：讓 Federate 決定是否讓 RTI 介入其內部運作的功能

HLA 規格允許 Federate 在執行過程動態改變資料傳送方式，因此我們也將此功能包裝成易懂的名稱。


Federate A
RTI
[AttributeRelevanceAdvisory]
enableAttributeRelevanceAdvisorySwitch()
Set AttributeUpdateAdvisory = ON/OFF
disableAttributeRelevanceAdvisorySwitch()
[AttributeScopeAdvisor]
enableAttributeScopeAdvisorySwitch()
Set AttributeScopeAdvisory = ON/OFF
disableAttributeScopeAdvisorySwitch()
[ClassRelevanceAdvisory]
enableClassRelevanceAdvisorySwitch()
Set ClassUpdateAdvisory = ON/OFF
disableClassRelevanceAdvisorySwitch()
InteractionRelevanceAdvisory
enableInteractionRelevanceAdvisorySwitch()
disableInteractionRelevanceAdvisorySwitch()

D、Data Distribution Management

接著的是 Data Distribution Management 的部份，我們由圖 3-25 可以知道分為 Region control、Object 以及 Interaction，分別是處理資料影響區域設定以及 Object 的 DDM 與 Interaction 的 DDM。

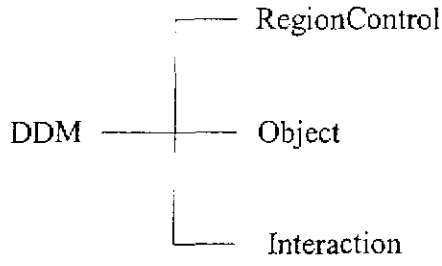


圖 3-25

再經由表 3-18，更進一步的發現可以提出 theRegion 這個參數，但是這個參數以及整個 DDM 的執行都啟動於 Region control，因此我們將 Region control 的功能以及 theRegion 這個參數放置於這個 Class 中。

	DDM	RegionControl	Object	Setting	Associat	General	Scope	Self
CreateRegion()								numberOfExtents
deleteRegion()								
modifyRegion()								numberOfExtents
notifyAboutRegionModification()		theRegion(theSpace)						
registerObjectInstanceWithRegion()								theObject theAttribute
requestClassAttributeValueUpdateWithRegion()								theAttribute
subscribeObjectClassAttributesWithRegion()								attributeList active
unsubscribeObjectClassWithRegion()				theClass theRegion				
associateRegionForUpdates()								theAttribute
UnassociateRegionForUpdates()			theRegion		theObject theRegion			
SendInteractionWithRegion()								theInteraction theParameters theTime theTag
subscribeInteractionClassWithRegion()								active
unsubscribeInteractionClassWithRegion()	theRegion					theRegion theClass		

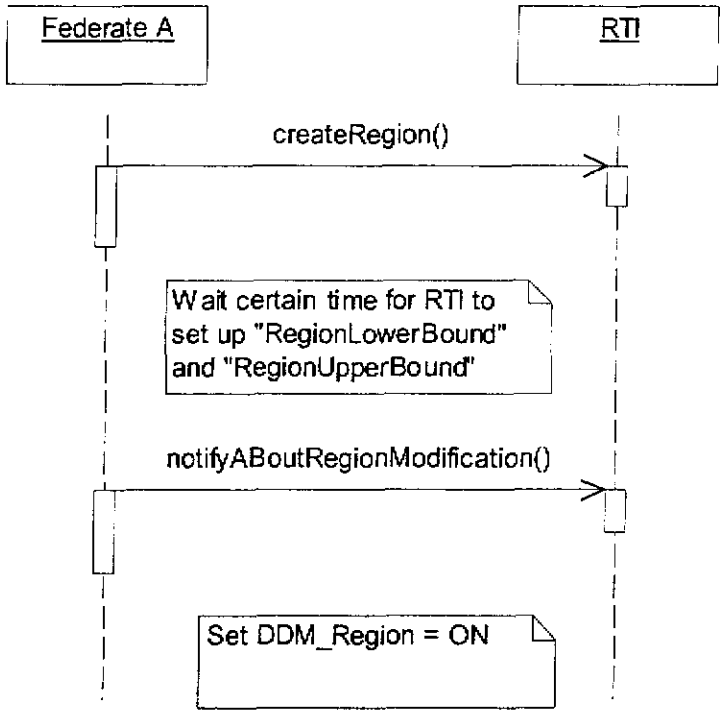
表 3-18

在此 Class 中包含有啟動這個模組功能的開關，並在選用開啟同時設定 routing space。由於我們已將此類別與 Object 及 Interaction 有關的 service call 與

Object Management 相關的 service call 結合，因此，這個 class 只需包含建構子、解構子以及一些工具 Function 的執行功能即可。

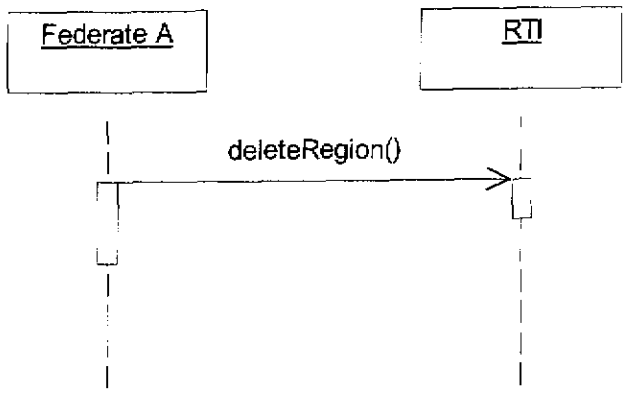
Constructor：Data Distribbution Management 的建構子

在這個 class 中我們經由建構子來設定 Routing Space，並以此建構子當中的一個參數當做開關，如果開啟 Data Distribution 的機能，則所有有關 DDM 的 service call 將會可被使用。



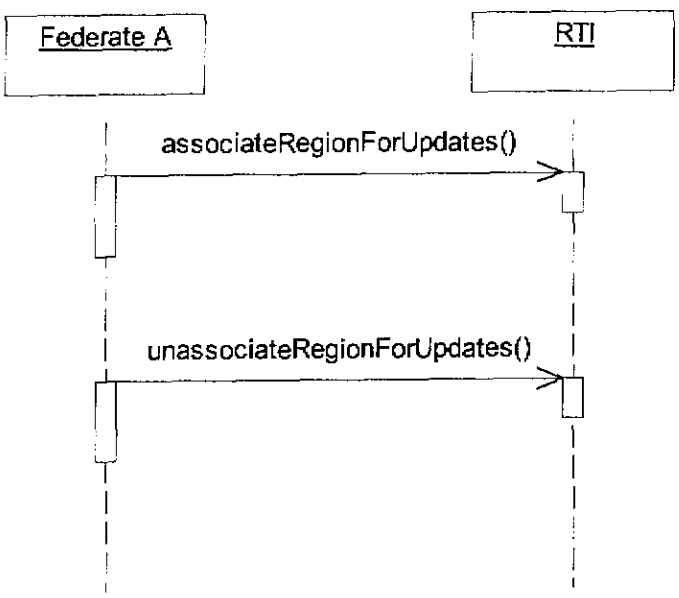
Constructor：Data Distribbution Management 的解構子

解構子功能在於將設定好的 Routing Space 取消。



Associate：用來指定哪些物件資料可以用資料影響範圍的方式來傳送

這個 Function 的適用來動態設定某個物件的資料影響範圍(Routing space)，此功能提供一個可以在執行過程動態更改 Routing space 的工具。



E、Time Management

Time Management 部份我們可以由圖 3-19 來分析，可以分為三大類，分別是 Federation、Interaction 與 Object。因此此類別的 class 可分為一個 base calss 及三個繼承的 sub-classes，分別屬於 Federation、Interaction 與 Object。其中 Interaction 與 Object 這兩個較簡單，只是改變資料傳送方式而已。

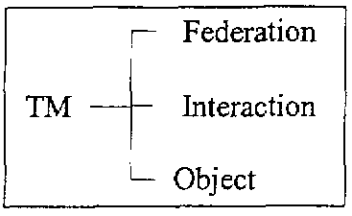


圖 3-19

而在 federation 這個部份，由圖 3-20 我們可以看到三個節點，分別是 Switch、Advance、Queries。

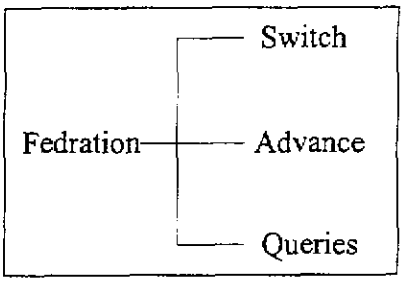


圖 3-20

而 Queries 中的介面函式為與 Federation 有關的基本工具組，因此我們放置於 Federation 的繼承 sub-class 內。另外，Advance 與 Switch 兩個節點則與 Federate 內所有物件與事件有關係的服務，因此將這些介面函式放置於 Time Management 的 base class 中。Advance 是關於在模擬過程當中，Federate 要球整個模擬時間推進的方式，而由圖 3-22 得知，HLA 規格所定義的時間推進方式可分為 Time Step，Event，與 Optimistic 三種方式，因此我們也必須個別給予 functions 來使用。

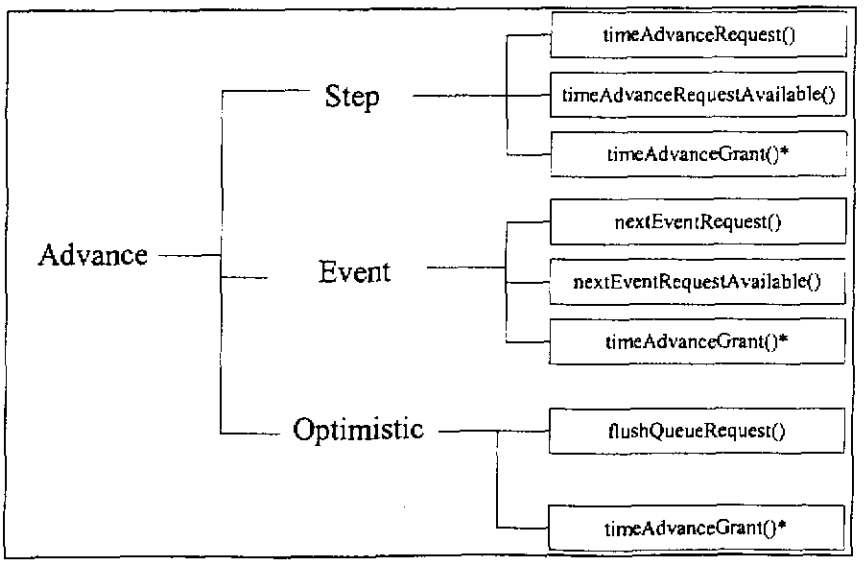
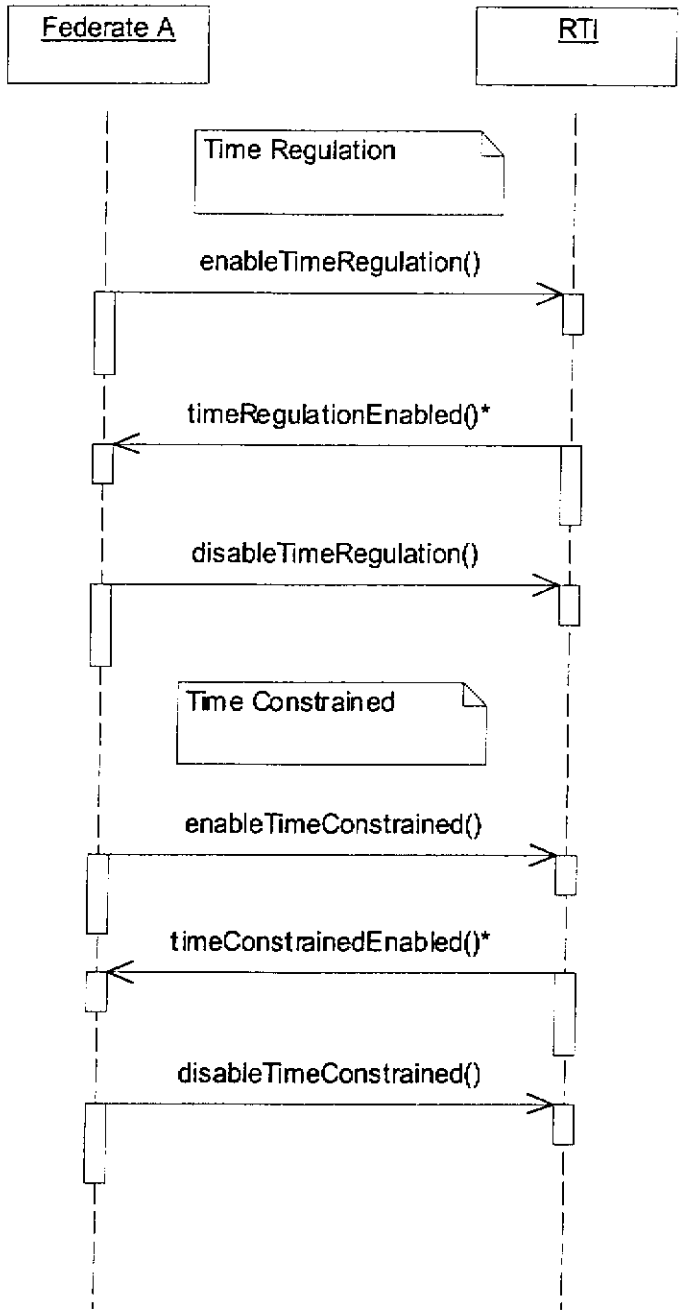


圖 3-22

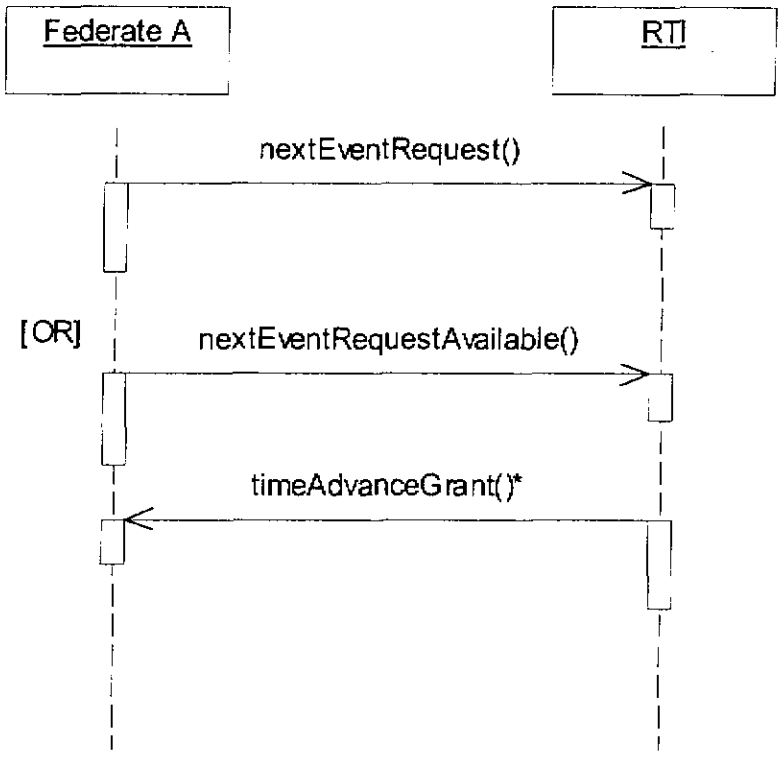
Constructor：Time Management 的 base calss 建構子

這個 Function 以 switch 的方式來選擇設定資料接收與傳輸的方式。



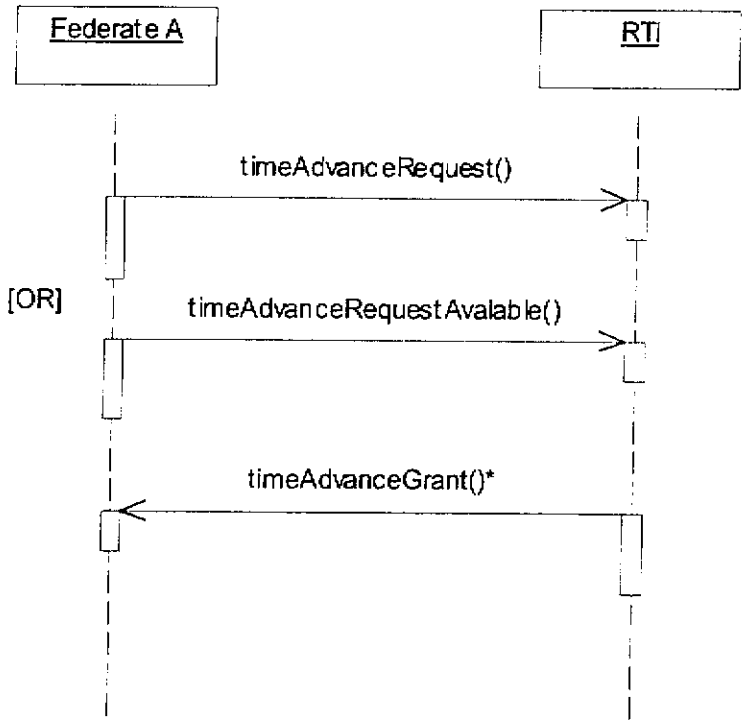
EventTime：要求邏輯時間以事件時間為推進基準

此 function 是以下一個 time-stamp-ordered 事件為基準來推進 federate 的邏輯時間至該事件的時間點。兩種呼叫方式的差別是在於使用 nextEventRequestAvailable()此功能可能不會等到時間點之前的所有事件皆被處理完成就會推進 federate 的邏輯時間，因此不能百分之一百保證時間的一致性。



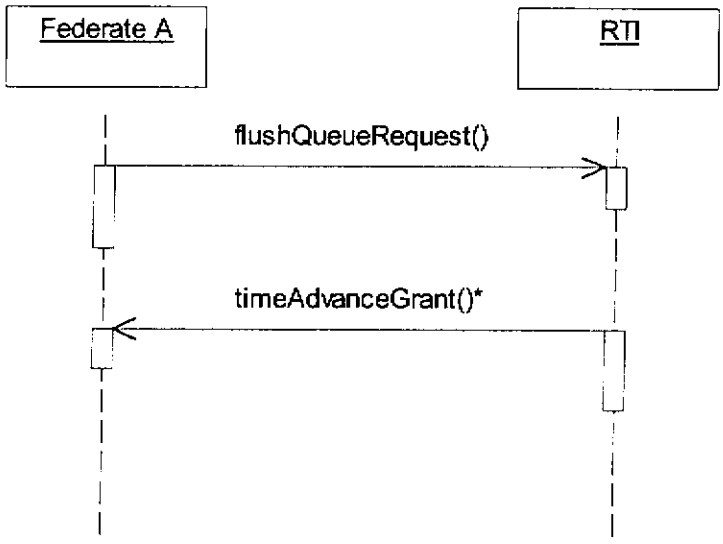
TimeStep：要求指定的邏輯時間為推進基準

此 function 是輸入的指定時間為基準來推進 federate 的邏輯時間至該時間點。兩種呼叫方式的差別是在於使用 timeAdvanceRequestAvailable()此功能可能不會等到時間點之前的所有事件皆被處理完成就會推進 federate 的邏輯時間，因此不能百分之一百保證時間的一致性。



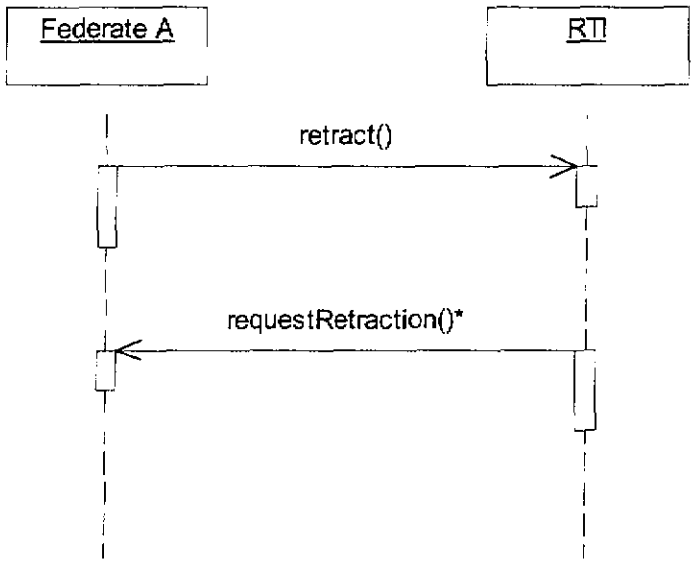
Optimistic：加快邏輯時間的推進

此 function 是 federate 內所有 event queue 的所有事件全部送出(不管是否會違反時間邏輯關係)，使得 federate 的邏輯時間可以儘早推進。



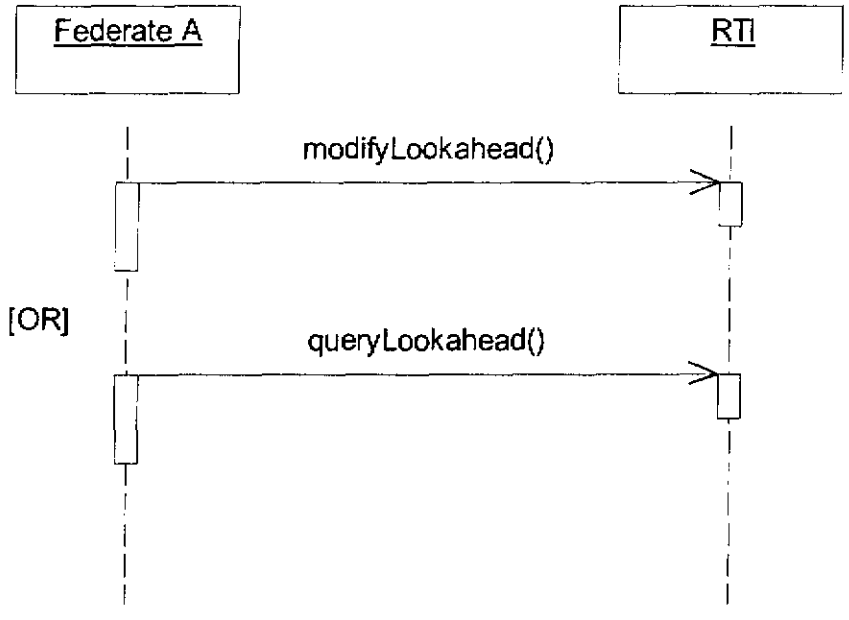
Retract：取消所有的排定的工作

此 function 會將所有按照時間次序排定的工作完全取消，包括物件資料與事件訊息的傳送。



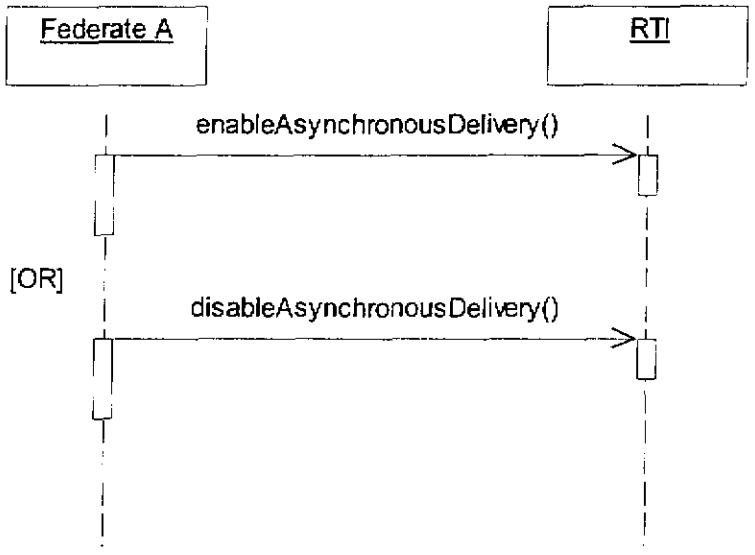
Lookahead：邏輯時間推進間隔的相關函式

HLA 規格內邏輯時間的推進是依據指定的時間間隔來推進，此 function 綜合動態更改時間推進間隔的功能及查詢現有的時間推進間隔的功能。



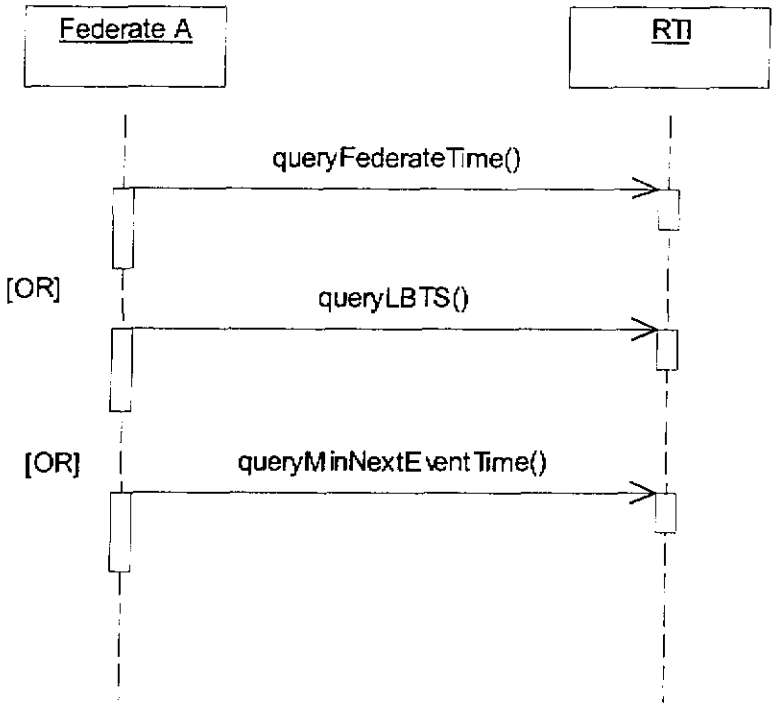
Delivery：Federate 通知 RTI 可以開始傳送訊息

此 function 通知 RTI 可以開始傳送 receive-ordered 的事件，不必等待時間推進功能是否有啟動。



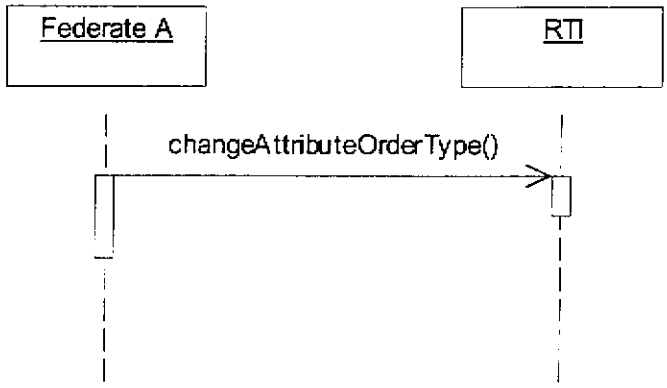
Query：與 Federation 有關的 Time Management 功能

此 function 讓 federate 可以向 RTI 查詢 federation 的邏輯時間相關資訊，包括目前的邏輯時間，最小時間推進間隔，及下一個事件的最早發生時間。



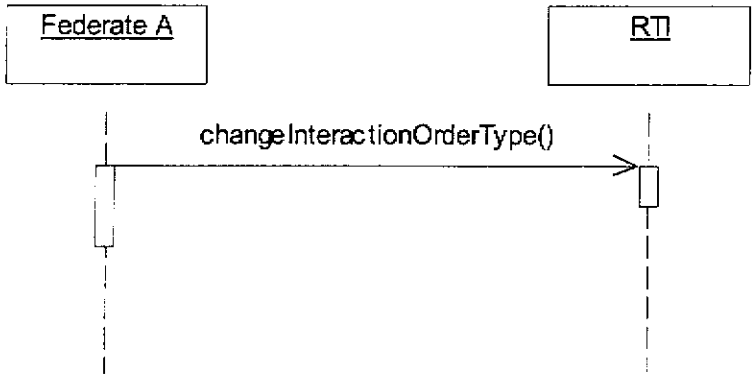
ChangeOrder：與 Object 有關的 Time Management 功能

此 function 讓 federate 可以向 RTI 要求更改物件屬性資料的傳送次序。



ChangeOrder：與 Inteaction 有關的 Time Management 功能

此 function 讓 federate 可以向 RTI 要求更改互動事件資訊的傳送次序。



F、Ownership Management

針對 Ownership Management 的部份，我們由執行的功能可以發現目標都是 Object，再經由查詢圖 3-14 發現在 Ownership Management 中除了 Query 節點下的函式為這個 class 的基本工具外，其餘三個節點是以 Ownership 的執行方法來分類，因此都可以是所定義之 class 的 member function。因此整個 Ownership Management 只需要是一個 class 即可。

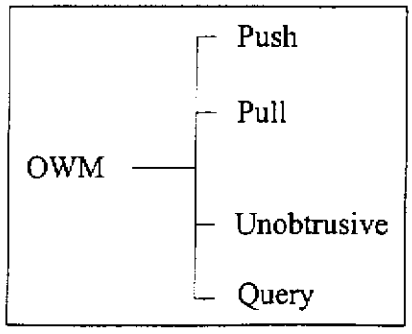
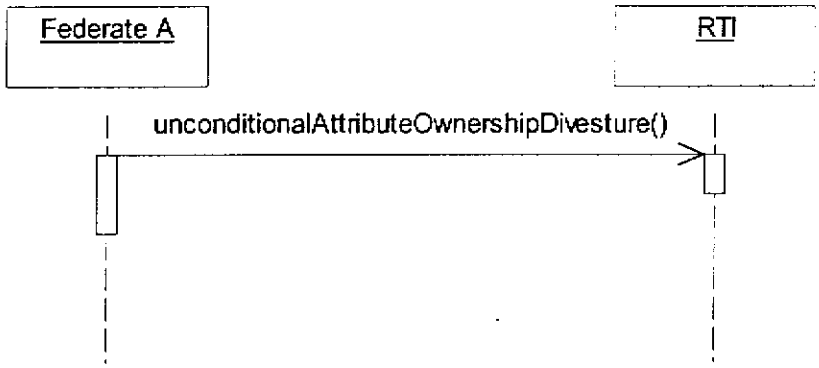


圖 3-14

這些 Function 的執行方式因為分為三大類型，分別是無條件放棄(Abandon)、謙卑協商式轉移(Unobtrusive)、主動放棄(Push)、被動放棄(Pull)等四種不同的 ownership 轉移方法，分別詳述如下。

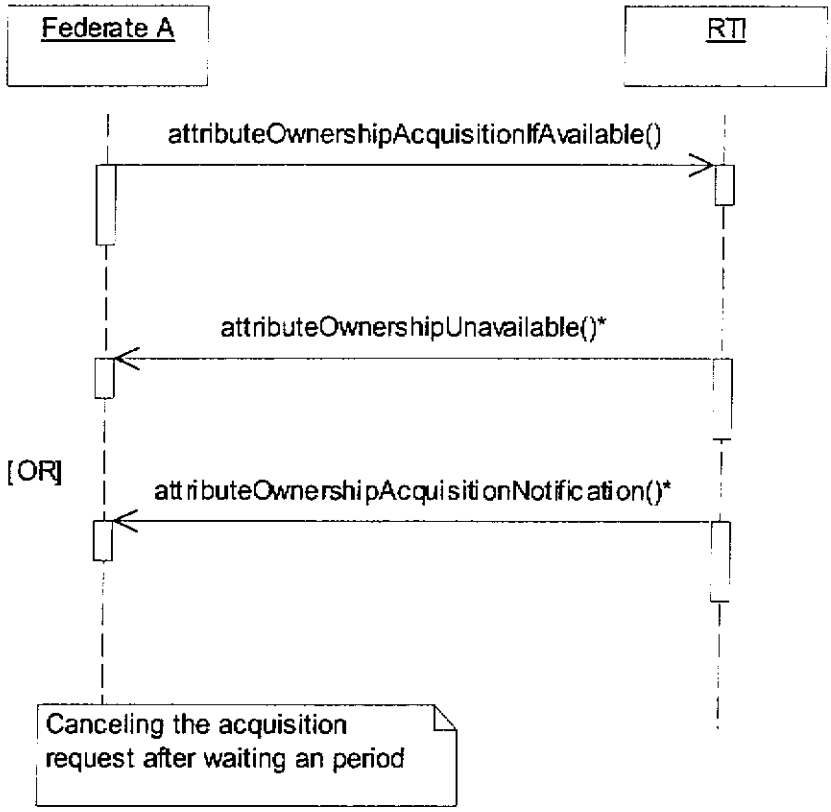
Abandon：Federate 無條件放棄擁有控制權

Federate 使用此 function 來主動放棄某個物件資料屬性的控制權，此時將由 RTI 接收此物件資料屬性。



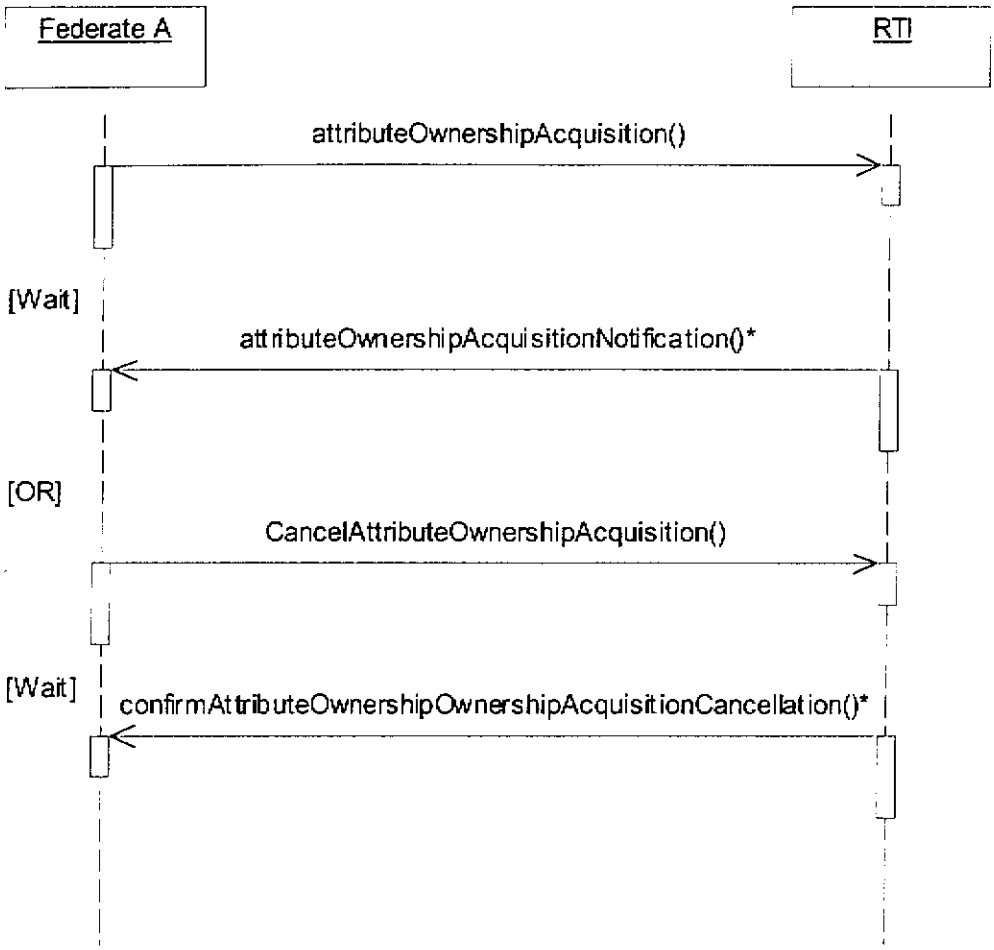
Unobtrusive：Federate 以謙卑的協商方式要求控制權轉移

Federate 謙卑的詢問 RTI 關於某個物件資料屬性的控制權可否取得？若此物件屬性目前無法移轉，則 RTI 會以 callback 方式回覆 attributeOwnershipUnavailable，否則 RTI 開始進行控制權移轉工作。



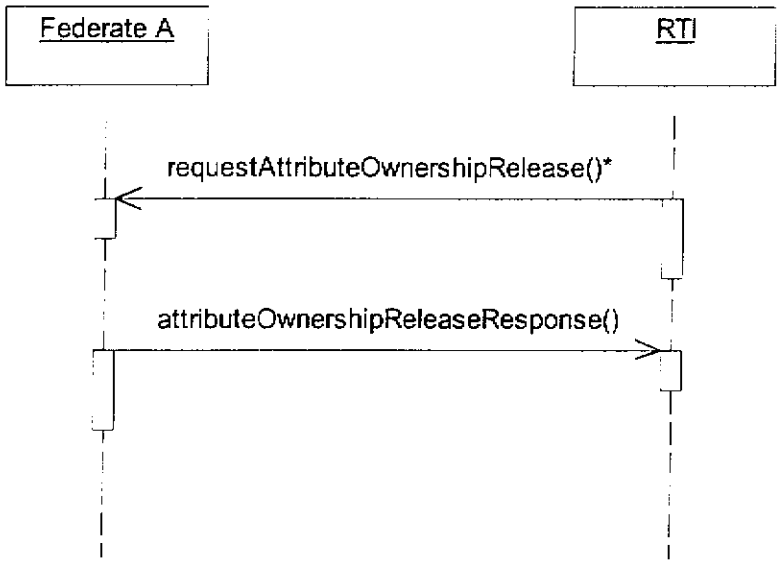
Request：Federate 以 pull 方式要求控制權轉移

Federate 向 RTI 要求取得某個物件資料屬性的控制權，然後等著 RTI 通知它取得控制權。Federate 也可以用此 function 向 RTI 取消此項要求，而 RTI 會向他確認取消動作。



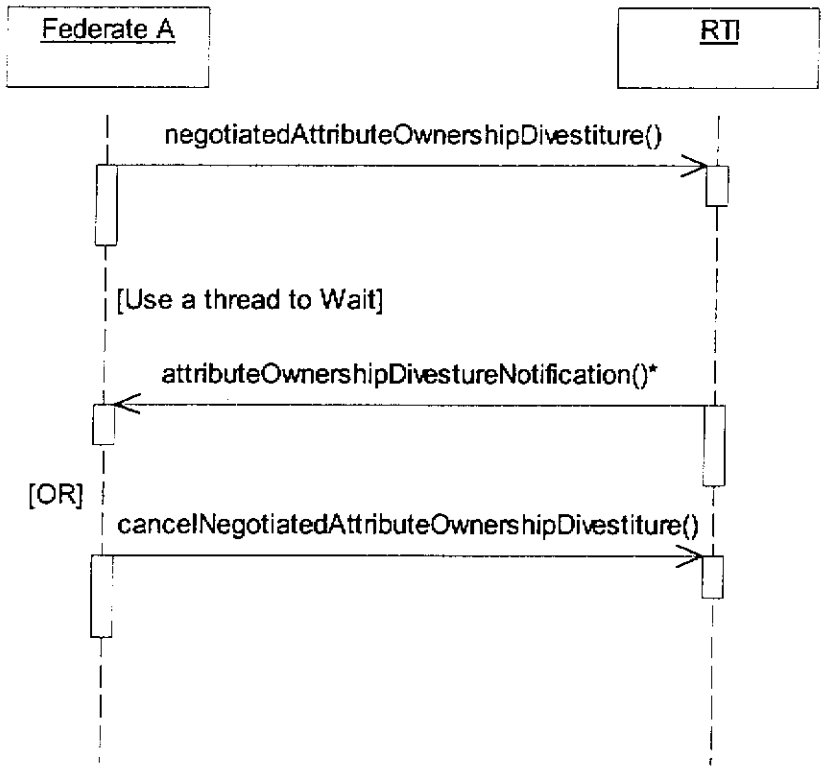
Relinquish：Federate 被要求放棄控制權

Federate 被 RTI 強迫放棄某個物件資料屬性的控制權，Federate 以 attributeOwnershipReleaseResponse 向 RTI 轉移控制權。



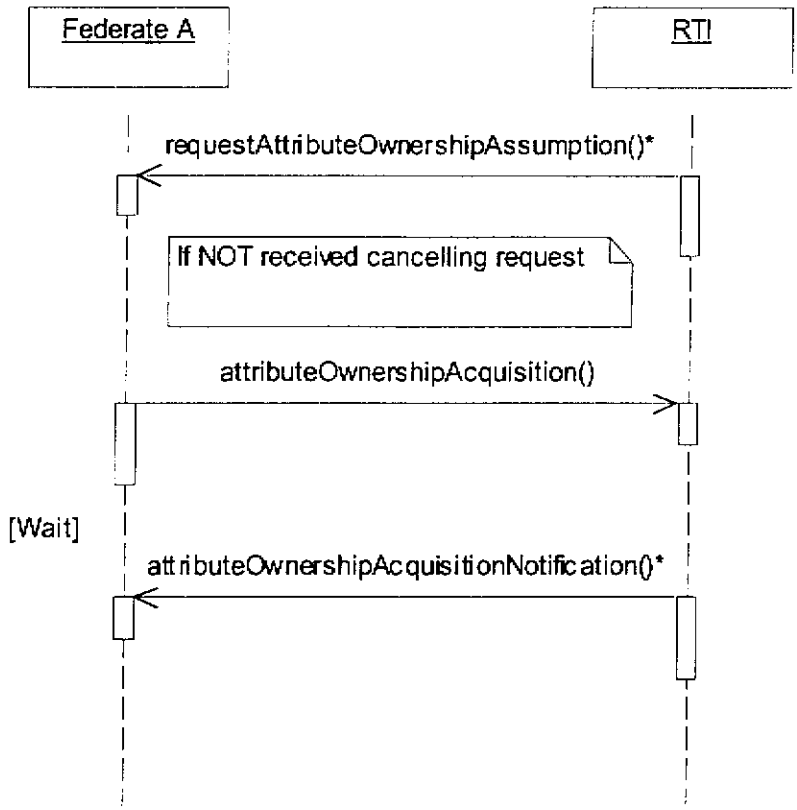
Discard：Federate 主動放棄擁有控制權

Federate 主動向 RTI 表示要放棄某個物件資料屬性的控制權，交由 RTI 去詢問是否有任何 Federate 對此物件資料屬性的控制有興趣？



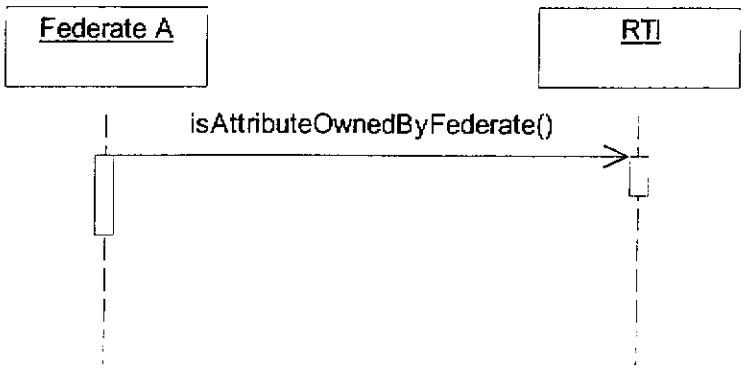
Accept：Federate 接受某個 Federate 所主動放棄的控制權

Federate 接受 RTI 的詢問，表示要接受某個物件資料屬性的控制權，RTI 會先以 requestAttributeOwnershipAssumption 來詢問 Federate，而 Federate 則以 attributeOwnershipAcquisition 來表示意願，最後 RTI 會以 attributeOwnershipAcquisitionNotification 來告知 Federate 可開始控制該物件資料屬性。



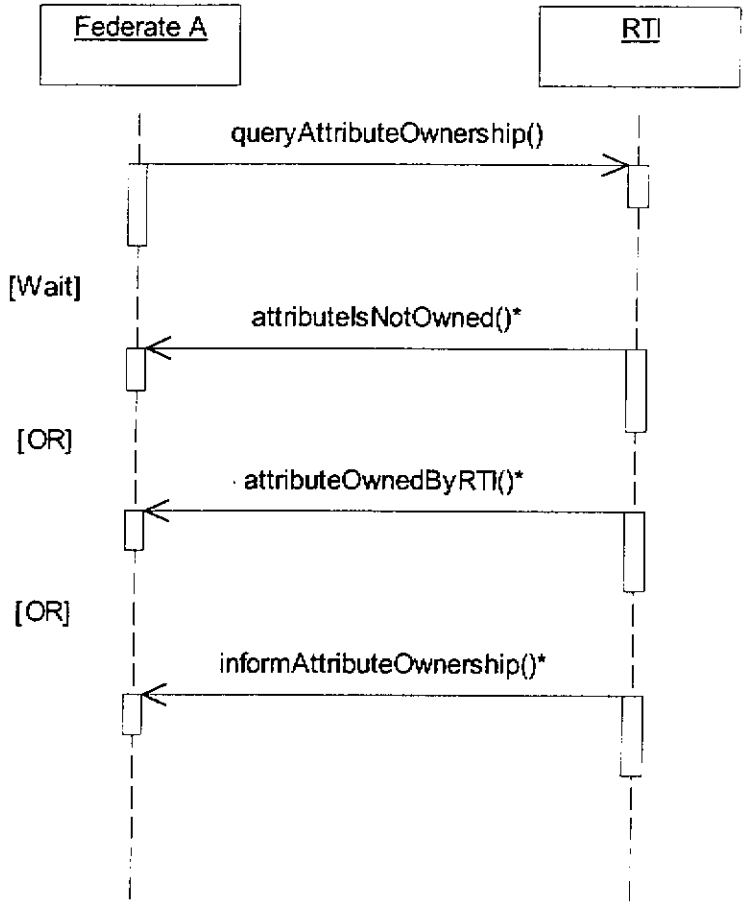
QueryLocal：Federate 詢問 RTI 是否確實擁有某個物件屬性的控制權

Federate 使用此功能來詢問 RTI 其是否仍擁有某個物件屬性的控制權？



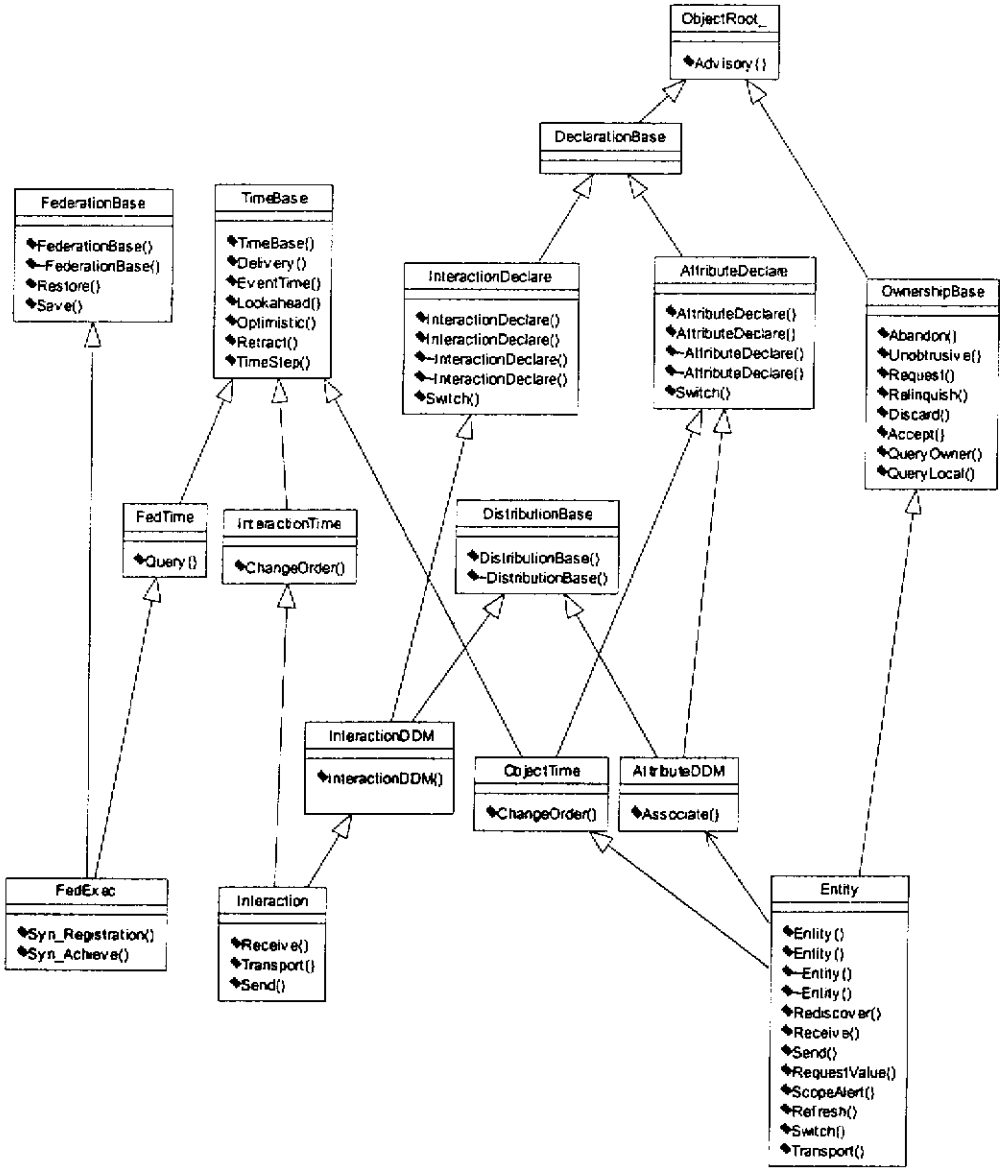
QueryOwner：Federate 詢問 RTI 誰擁有某個物件屬性的控制權？

Federate 使用此功能來詢問 RTI 某個物件屬性控制權的擁有者。RTI 的回答有三種可能性，其一是該屬性不為任何人所擁有，其二是該物件屬性為 RTI 所擁有，最後是告知擁有該物件屬性的 federate 資訊。



第二部份、類別程式庫階層架構

藉由第一部份的功能函式分析與定義，我們已經可以清楚得知 HLA 規格所欲達到的功能目標，我們並藉此定義出個別的功能函式，於是我們接著可以再次將 3.2 節的參數分析結果提出與第一部份所定義的功能函式作交互比對，釐清出哪些功能函式應該包裝在相同類別內，而哪些包裝完成的類別之間具階層關係，進而架構出如下的類別程式庫階層。



以下我們還是依照 HLA 規格的功能服務模組，以及第一部份的分析定義步驟來一一說明類別程式庫階層的建立過程。

A、Federation Management

在第一部份此功能模組的分析過程中，我們得知 Federation Management 的呼叫函式只有 Synchronization Point 的定義與使用對 Federation 的執行過程有直接影響，其餘的屬於一般化的初始及結束步驟，因此我們為 Federation Management 包裝出一個稱為 FedeationBase 的基礎類別模組，其下有一個直接衍生出的類別，從模擬啟始階段就會存在一直到結束，而這個 class 稱為 FedExec，如圖 3-31 所示。

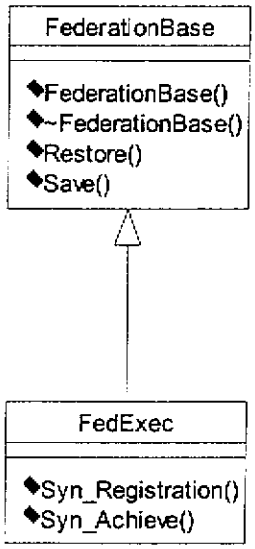


圖 3-31

B、Declaration Management

包裝完 Federation Management 後，我們接下來對圖 3-30 中 federation execution cycle 的第二個順序的 Declaration Management 包裝，在 Object 與 Interaction 創造實體(instance)與執行前，一定要經過宣告屬性的動作，因此，我們由表 3-5 到表 3-9 做交互比對，及前面一節的分析發現 Declaration Management 的定義必須先有一個基礎類別，我們稱為 DeclarationBase。其下在衍生出兩個子類別，分別為 InteractionClare 及 AttributeClare。在此我們之所以用 Attribute 來定義類別，而不是 Object，是因為 HLA 規格的所有宣告是以物件屬性為單位，而非物件個體。因此，我們可以進一步整合成下圖 3-32。

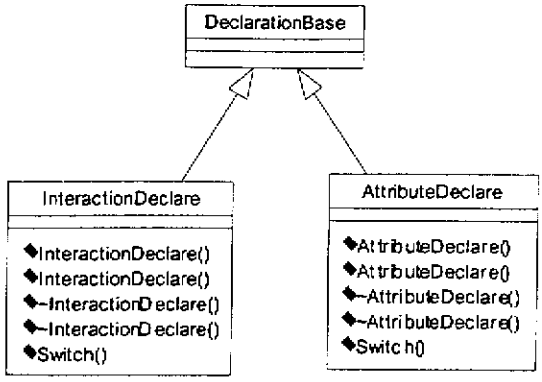


圖 3-32

C、Data Distribution Management

在包裝 Object Management 之前，由於 HLA 規格提供 Data Distribution Management 的宣告功能，而此宣告功能會影響 Object Management 的執行動作，因此我們接著必須先包裝 Data Distribution Management 類別。與 Declaration Management 相同，Data Distribution Management 的類別包裝也要分為 Object 與 Interaction，因此我們先定義一個稱為 DistributionBase 的基礎類別，然後衍生出兩個子類別，分別為 InteractionDDM 與 AttributeDDM。其次，由於 Data Distribution Management 與 Declaration Management 功能類似，只是增加條件性限制，因此我們讓 Data Distribution Management 的類別繼承自 Declaration Management，如圖 3-33 所示。

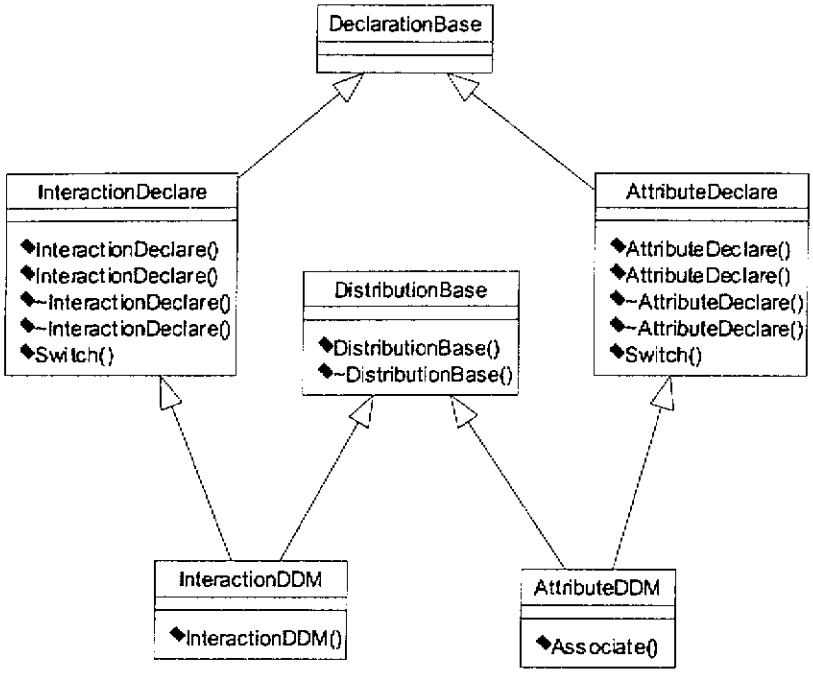


圖 3-33

D、Object Management

有了上述完整的宣告，我們就可以開始包裝 Object Management，由前一節的分析可以得知，Object Management 的函式呼叫中，除了 Advisory 相關函式呼叫外，其它的函式呼叫直接隸屬 Object 或 Interaction 的動作，因此我們一樣將 Object Management 類別分為 ObjectRoot_的基礎類別及 Interaction 與 Entity 兩個衍生類別。ObjectRoot_的基礎類別包含 Advisory 的所有函式呼叫，而由於 Advisory 相關函式呼叫與 Declaration management 及 Data Distribution Management 的宣告有關，因此我們將此基礎類別設計為類別階層的最基礎類別讓 DeclarationBase 類別繼承，而讓 Interaction 這個衍生類別繼承自 InteractionDDM 類別，至於 Entity 類別因為是以物件為單位，因此此類別是引用 AttributeDDM 類別，而非直接繼承，如圖 3-34 所示。

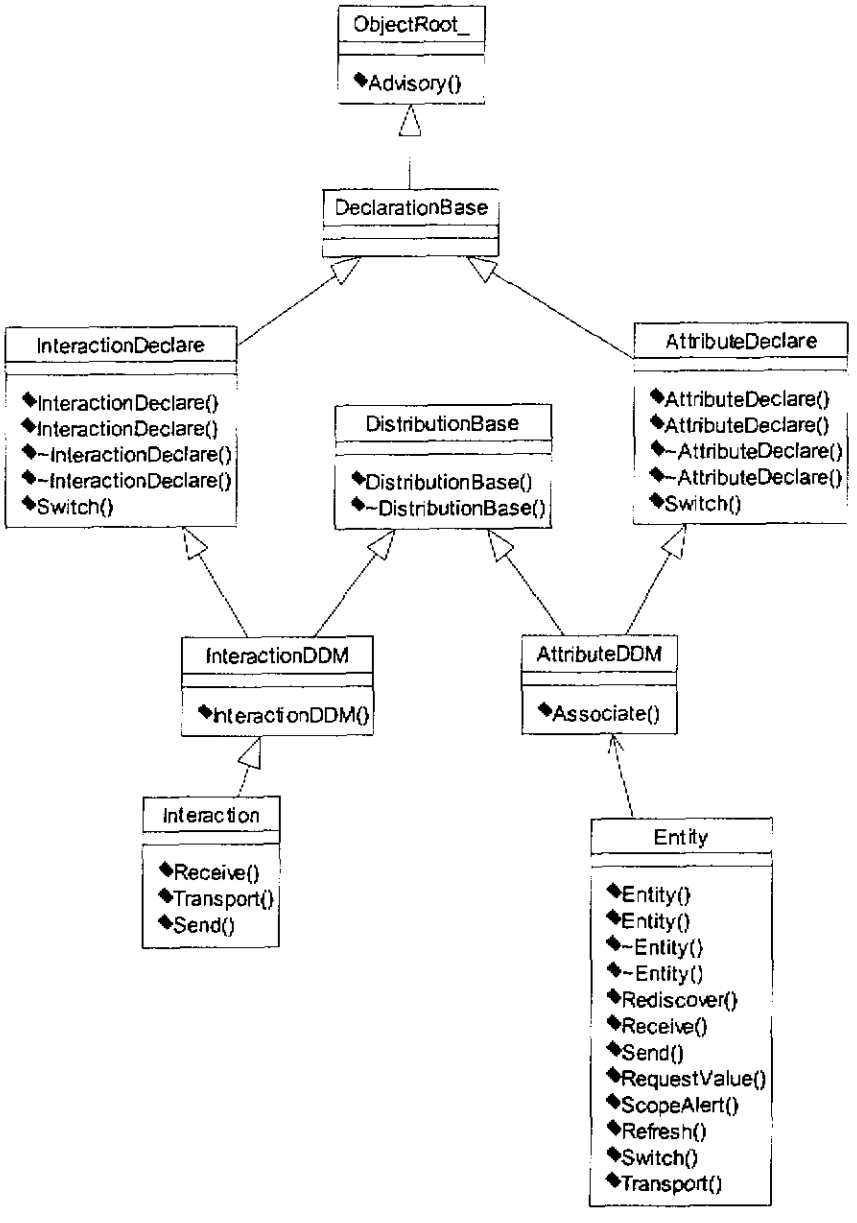


圖 3-34

E、Time Management

在 Time Management 方面，由前一節的分析得知此類別的 class 可分為一個基礎類別稱為 TimeBase 及三個繼承的 sub-classes，分別為 FedTime、InteractionTime 與 ObjectTime 三個衍生類別。其中，由於 ObjectTime 類別是會改變 DeclarationManagement 的宣告結果，因此將此類別以多重繼承的方式，同時也繼承自 AttributeDeclaration 類別，如圖 3-35 所示。

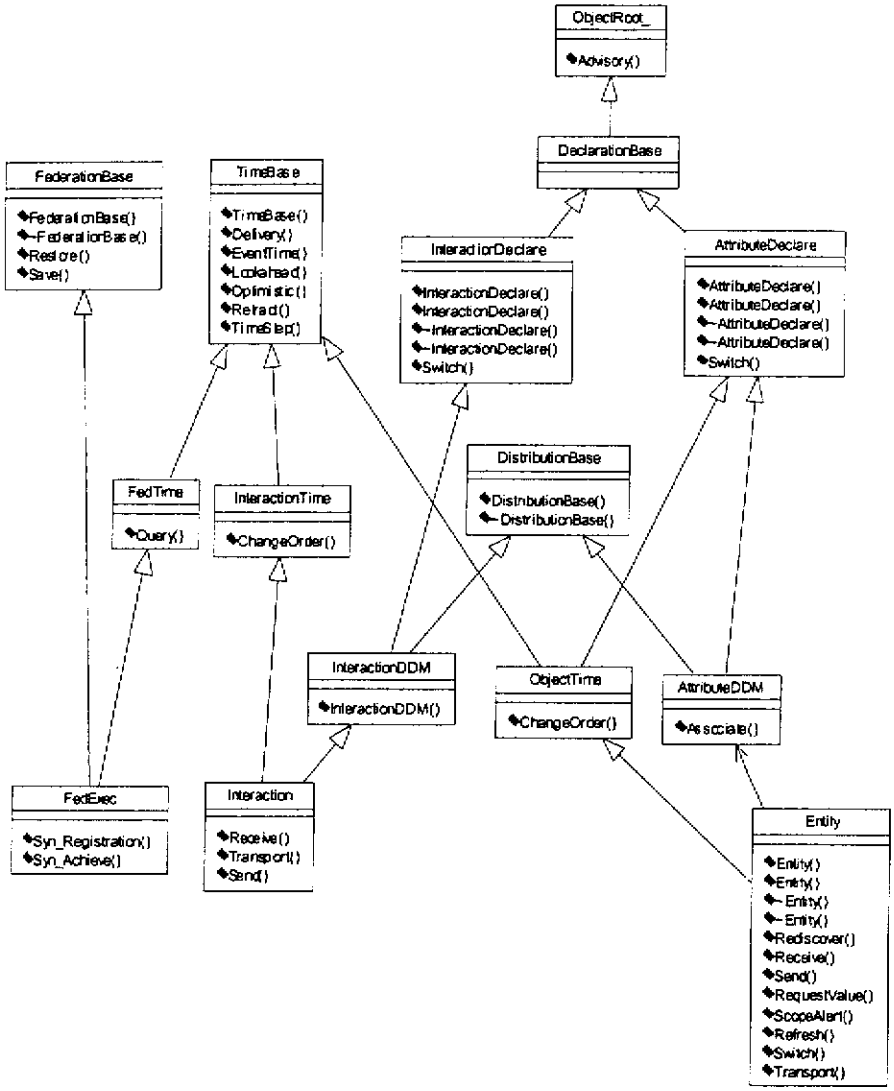


圖 3-35

F、Ownership Management

最後是 Ownership Management 的類別，由於此類別皆是針對物件屬性，因此我們直接將此類別包裝成一個類別讓 Entity 類別繼承，成為如圖 3-29 所示的完整類別資料庫階層圖。

第四章、實作規格

由第三章的分析結果，我們可以包裝出最淺顯易懂、容易使用的程式庫，讓 federation 設計者可以只以簡單的程序建構出 federate 的程式，以及 federation 的主體。因此本章當中，將列出包裝後的程式庫，詳細的說明 class 內容、使用時機及各個 Function 的功能。第三章所分析結果完全是依照 HLA 規格加以分析包裝，而本章將說明實作結果，由於分析結果在實作過程中必須稍作細部調整以配合實際環境，例如加上 FederateAmbasaddor 及 FedAgent 等類別物件的繼承，如圖 4-1 所示，為本章內容為計劃的實作結果。

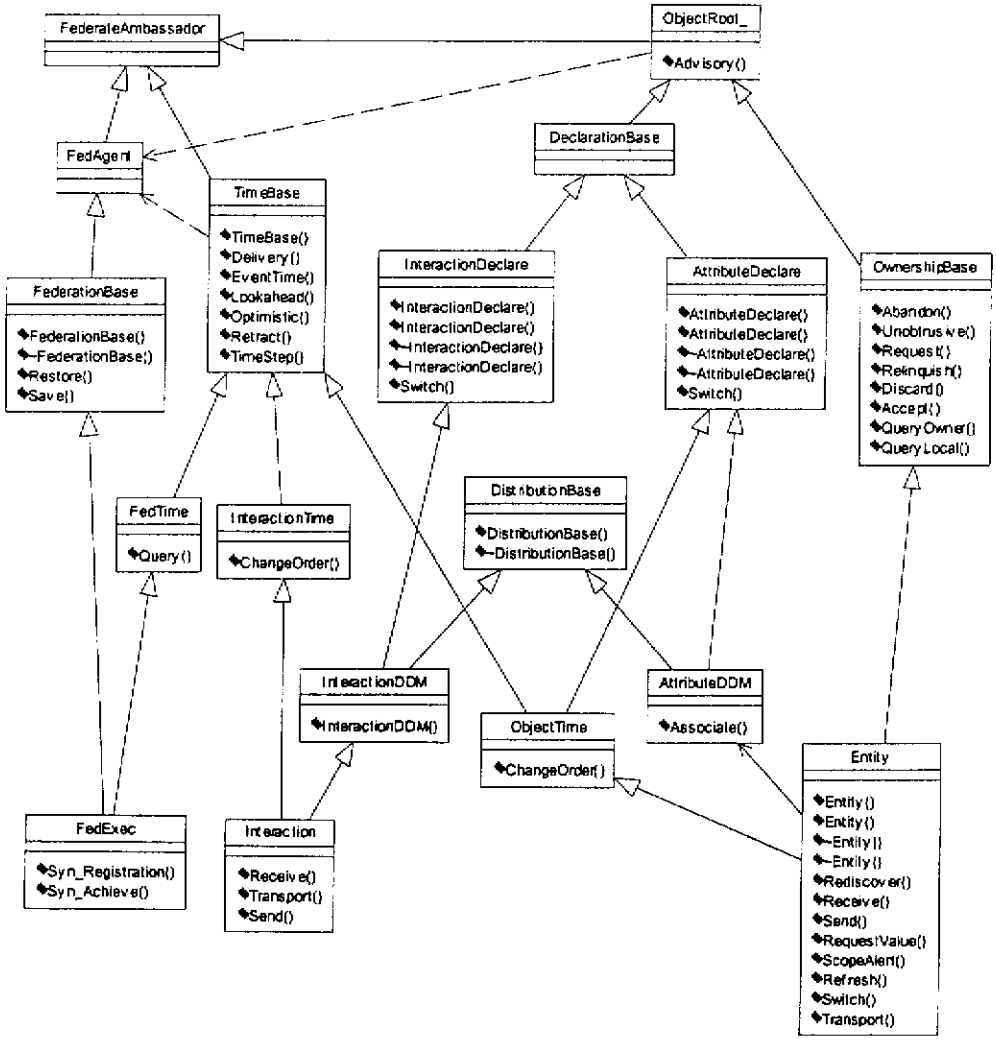


圖 4-1 RTI 程示庫整體架構圖

如圖 4-1 所示，由於 HLA 規格使用 callback function 方式使得 RTI 可以主動通知 Federate 有關模擬方面的訊息，而在 C++的實作規格是無法將此 callback function 實作在物件類別內，因此我們設計一個 FedAgent 類別來繼承 FederateAmbassador，而將 HLA 規格所有的 callback functions 實作在該 FedAgent 類別內，使得整個 Federate 只有一個 FedAgent 物件用來接收 RTI 主動通知的訊息，然後呼叫相關的模擬物件以接收 RTI 訊息。

此外，本計劃實作之類別程式庫使用如下的代號，以方便辨認即以後使用：

1. 在每個 class name(類別名稱)最前面，我們都加上 tv 字首作為以 HLA 規格及其他類別庫區分的符號，而每一個英文單字第一個字母為大寫，其餘字母為小寫。

2. Class Data Member (類別資料成員)的第一個英文單字全部為小寫，之後每一個英文單字的第一個字母為大寫，其餘字母為小寫。

3. Class Member Function (類別成員函式)的每一個英文單字的第一個字母為大寫，其餘字母為小寫。

4. 在每個 Function Parameter (函式參數)的最前面以'_'底線符號為字首，第一個英文單字全部為小寫，之後每一個英文單字的第一個字母為大寫，其餘字母為小寫。

第一部份、Class Library

A. Class Name : tvAttributeDDM

Description：

tvAttributeDDM Class 主要用來建立與區域相關聯的物件屬性實體。

Synopsis：

#include <tvAttributeDDM.h>

class tvAttributeDDM;

Data Member：

None

Functional Member：

Constructor function:

A.1 tvAttributeDDM(tvFedAgent* _agent, tvRTIObjectHandle _handle,tvItemSet& _publish,tvItemSet& _subscribe);

General functions:

A.2 void Associate(bool _associate);

B. *Class Name : tvAttributeDeclare*

Description：

tvAttributeDeclare Class 主要用來組織物件的屬性架構。

Synopsis：

#include < tvAttributeDeclare.h>

class tvAttributeDeclare;

Data Member：

bool registration;

儲存 Object Registration Switch 旗標。

Functional Member：

Constructor function:

B.1 tvAttributeDeclare(tvFedAgent * _agent,tvItemSet& _publish, tvItemSet& _subscribe);

Destructor function:

B.2 ~tvAttributeDeclare();

Callback functions:

B.3 virtual void Switch(tvRTIObjectClassHandle theClass,bool registration);

C. Class Name : tvDeclarationBase

Description：

tvDeclarationBase Class 主要包含 Declaration management 所需要的基礎元件。

Synopsis：

#include < tvDeclarationBase.h>

class tvDeclarationBase;

Data Member：

None

Functional Member：

Constructor function:

C.1 tvDeclarationBase(tvFedAgent * _agent);

D. Class Name : tvDistributionBase

Description：

tvDistributionBase Class 主要用來建立資料散佈範圍的物件，並與其屬性(Attribute)或互動(Interaction)衍生類別的實體產生關聯。

Synopsis：

#include < tvDistributionBase.h>

class tvDistributionBase;

Data Member：

tvRTIRegion* region;

資料散佈範圍的物件。

Functional Member：

Constructor function:

D.1 tvDistributionBase(tvItemSet& regionDef);

Destructor function:

D.2 virtual ~tvDistributionBase();

E. Class Name : tvEntity

Description：

tvEntity Class 主要用來建立模擬環境中 Avatar 的資料實體，相當於 HLA 環境中的 Object，此類別包含更新與接收狀態資料的函式。

Synopsis：

#include <tvEntity.h>

class tvEntity;

Data Member：

tvRTIObjectHandle handle;

模擬環境中唯一的物件 ID。

string name;

此實體的名稱。

Functional Member：

Constructor function:

E.1 tvEntity(tvFedAgent* _rtiBase,const char* _name);

Destructor function:

E2. virtual ~tvEntity();

General functions:

E3. void Rediscover();

E4. void RequestValue(tvRTIAttributeHandleSet& theAttributes);

E5. void Send(tvRTIAttributeHandleValuePairSet* _set , tvRTIFedTime theTime);

E6. void Send(tvRTIAttributeHandleValuePairSet* _set);

E7. void Transport(tvRTIAttributeHandleSet& _theAttributes,tvRTITransportationHandle _theType);

Callback functions:

E8. virtual void Receive(tvRTIAttributeHandleValuePairSet& _set);

E9. virtual void Refresh(tvRTIAttributeHandleValuePairSet& _set);

E10. void ScopeAlert(bool _scope , tvRTIObjectHandle _handle , tvRTIAttributeHandleSet& _set);

E11. virtual void Switch(tvRTIAttributeHandleValuePairSet& theAttributes , bool _switch);

F. Class Name : tvFedAgent

Description：

tvFedAgent Class 主要用來繼承 tvFederateAmbassador 類別(即 RTI 中的 RTI::FederateAmbassador 類別)，以接收所有來自 RTI 的事件，並分別的把接收到的事件傳送給適當的實體(Federate、Object、Attribute 以及 Interaction)。

Synopsis：

#include < tvFedAgent.h>

class tvFedAgent;

Data Member：

tvRTIAmbassador rtiAmb;

RTI Ambassador 的實體，透過它呼叫 RTI 的 API 函式。

Functional Member：

Callback functions:

與 RTI::FederateAmbassador Class 的函式成員相同。

G. Class Name : tvFederationBase

Description：

tvFederationBase Class 主要用來建立可提供 Federate 之間互動的模擬互動環境的 Federation，並提供儲存與回復 Federation 內部狀態的成員函式。

Synopsis：

#include < tvFederationBase.h>

class tvFederationBase;

Data Member：

None

Functional Member：

Constructor function:

G.1 tvFederationBase(const char* _executionName,const char* _federateName ,const char* _fed);

Destructor function:

G.2 virtual ~tvFederationBase();

General functions:

G.3 const char* GetExecutionName();

G.4 tvRTIFederateHandle GetFederateHandle();

G.5 const char* GetFederateName();

G.6 const char* GetRestoreLabel();

G.7 const char* GetSaveLabel();

G.8 bool IsFedExecCreator();

G.9 void Restore(const char* label);

G.10 void Save(const char* label);

G.11 void Tick();

G.12 bool Tick(double min,double max);

H. Class Name : tvFedExec

Description：

tvFedExec Class 用來繼承 tvFederationBase Class 以加強在 Federation 的功能，在此類別強調 Federate 之間的同步問題。

Synopsis：

#include < tvFedExec.h>

class tvFedExec;

Data Member：

None

Functional Member：

Constructor function:

H.1 tvFedExec(const char* _executionName,const char* _federateName ,const char* _fed);

General functions:

H.2 void Syn_Achieve();

H.3 bool Syn_Registration();

I. Class Name : tvFedTime

Description：

tvFedTime Class 繼承 tvTimeBase Class 所以具有 Time management 基本的元素，而此類別主要提供 Federate 查詢相關的模擬時間。

Synopsis：

#include < tvFedTime.h>

class tvFedTime;

Data Member：

None

Functional Member：

General functions:

I.1 tvRTIFedTime Query(int _queryKind);

J. Class Name : tvInteraction

Description：

tvInteraction Class 主要用來建立 Interaction 的實體。

Synopsis：

#include < tvInteraction.h>

class tvInteraction;

Data Member：

None

Functional Member：

General functions:

J.1 void Send(tvRTIParameterHandleValuePairSet& theParameters);

J.2 void Send(tvRTIParameterHandleValuePairSet& _theParameters , tvRTIFedTime _theTime);

J.3 void Transport(tvRTITransportationHandle _theType);

Callback functions:

J.4 virtual void Receive(tvRTIParameterHandleValuePairSet& _theParameters);

K. Class Name : tvInteractionDDM

Description：

tvInteractionDDM Class 可使衍生的 tvInteraction Class 能夠制定 Interaction 事件的資料散佈範圍。

Synopsis：

#include < tvInteractionDDM.h>

class tvInteractionDDM;

Data Member：

None

Functional Member：

Constructor function:

K.1 tvInteractionDDM(tvFedAgent* _agent, tvItemSet& _publish , tvItemSet& _subscribe, tvItemSet& regionDef , bool DDM_Region=false);

L. Class Name : tvInteractionDeclare

Description：

tvInteractionDeclare Class 主要用來組織 Interaction 的參數(Parameter)成員。

Synopsis：

#include < tvInteractionDeclare.h>

class tvInteractionDeclare;

Data Member：

bool InteractionSwitch;

紀錄此 Interaction 的傳送開關旗標，true 代表只少有一個 Federate 訂閱了此 Interaction，false 代表此 Federation 並無任何 Federate 訂閱此 Interaction。

Functional Member：

Constructor function:

L.1 tvInteractionDeclare(tvFedAgent* _agent , tvItemSet& _publish , tvItemSet& _subscribe);

Destructor function:

L.2 virtual ~tvInteractionDeclare();

Callback functions:

L.3 virtual void Switch(tvRTIInteractionClassHandle theHandle,bool interactionSwitch);

M. Class Name : tvInteractionTime

Description：

tvInteractionTime Class 包含與調整 Interaction 相關的操作函式。

Synopsis：

#include < tvInteractionTime.h>

class tvInteractionTime;

Data Member：

None

Functional Member：

General functions:

M.1 void ChangeOrder(tvRTIInteractionClassHandle _theClass,
tvRTIOrderingHandle _theType);

N. Class Name : tvItemSet

Description：

tvItemSet Class 提供描述類別的元素的工具類別，每一個元素皆以字串表示

Synopsis：

#include < tvItemSet.h>

class tvItemSet;

Data Member：

None.

Functional Member：

Constructor function:

N.1 tvItemSet(const char* _className);

General functions:

N.2 bool Add(const char* _name);

N.3 string GetClassName();

N.4 string GetItemName(int _index);

N.5 int Size();

O. Class Name : tvObjectRoot

Description：

tvObjectRoot Class 封裝了與資料交換有關的模式設定機制。

Synopsis：

#include < tvObjectRoot.h>

class tvObjectRoot;

Data Member：

tvRTIAmbassador* rtiAmbPtr;

RTI Ambassador 的指標，提供繼承此類別的衍生類別呼叫 RTI 的 API 函式。

Functional Member：

General functions:

O.1 void Advisory(int _switchKind,bool _switch);

P. Class Name : tvObjectTime

Description：

tvObjectTime Class 封裝了與 Object 有關的時間管理機制。

Synopsis：

#include < tvObjectTime.h>

class tvObjectTime;

Data Member：

None

Functional Member：

General functions:

P.1 void ChangeOrder(tvRTIObjectHandle _theObject, tvRTIAttributeHandleSet& _theAttributes, tvRTIOrderingHandle _theType);

Q. Class Name : tvOwnershipBase

Description：

tvOwnershipBase Class 封裝了所有 Ownership management 的所有機制。主要提供 tvEntity 所產生的物件進行狀態屬性的控制權轉移。

Synopsis：

#include < tvOwnershipBase.h>

class tvOwnershipBase;

Data Member：

None

Functional Member：

General functions:

Q.1 void Abandon(tvItemSet& _attrSet);

Q.2 void Discard(tvItemSet &_attrSet);

Q.3 void Request(tvItemSet &_attrSet);

Q.4 bool QueryLocal(tvItemSet &_attrSet);

Q.5 int QueryOwner(tvRTIAttributeHandle _handle);

Q.6 void Unobtrusive(tvItemSet &_attrSet);

Callback functions:

Q.7 virtual void Accept(tvRTIAttributeHandleSet *_theAttributes);

Q.8 virtual void Relinquish(tvRTIAttributeHandleSet* _theAttributes);

R. Class Name : tvTimeBase

Description：

tvTimeBase Class 封裝了時間管理的時間前進機制。

Synopsis：

#include < tvTimeBase.h>

class tvTimeBase;

Data Member：

None

Functional Member：

Constructor functions:

R.1 tvTimeBase(bool constrain);

R.2 tvTimeBase(bool constrain,tvRTIFedTime& theLookahead);

R.3 tvTimeBase();

General functions:

R.4 void Delivery(bool _asyn);

R.5 tvRTIFedTime EventTime(bool _available=false);

R.6 tvRTIFedTime Lookahead();

R.7 void Lookahead(tvRTIFedTime _modifyTime);

R.8 tvRTIFedTime Optimistic();

R.9 void Retract(tvRTIEventRetractionHandle _rh);

R.10 tvRTIFedTime TimeStep(bool available=false);

第二部份、Function

A.1 tvAttributeDDM::tvAttributeDDM

Description：

建構資料散佈範圍並且與特定 Object 的 Attribute 產生關聯。

Synopsis：

```
tvAttributeDDM::
    tvAttributeDDM(
        tvFedAgent* _agent,
        tvRTIObjectHandle _handle,
        tvItemSet& _publish,
        tvItemSet& _subscribe
    );
```

Input value：

- tvFedAgent* _agent ： 此 Federation 的 tvFedAgent 的物件指標。
- tvRTIObjectHandle handle：使用 _publish 或_subscribe 所建立的 Object handle。
- tvItemSet& _publish：描述將要公開的屬性結構。
- tvItemSet& _subscribe：描述將要訂閱的屬性結構。

Return value：

None.

Exception：

None.

A.2 tvAttributeDDM::Associate

Description：

設定特定的 Object 的屬性是否與資料散佈範圍的區域產生關聯。

Synopsis：

```
void
tvAttributeDDM::
    Associate(
        bool _associate
);
```

Input value：

- bool _associate：如果為 true 代表要與資料散佈範圍的區域產生關聯，而 false 則是關閉此關聯。

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIAttributeNotDefined：一個屬性控制被不合法使用在特定的的物件類別或者物件實例的內容。
- tvRTIInvalidRegionContext：企圖刪除一個仍與屬性實例有關聯的部位。
- tvRTIRegionNotKnown：一個 Region 物件不被確認為一個呼叫 getRegion 函數時所產生的合法 Region 物件。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。

- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

B.1 tvAttributeDeclare :: tvAttributeDeclare

Description：

公開_publish 所描述的屬性結構以及訂閱_subscribe 所描述的結構。

Synopsis：

```
tvAttributeDeclare::
    tvAttributeDeclare(
        tvFedAgent * _agent,
        tvItemSet& _publish,
        tvItemSet& _subscribe
);
```

Input value：

- tvFedAgent * _agent：此 Federation 的 tvFedAgent 的物件指標。
- tvItemSet& _publish：描述將要公開的屬性結構。
- tvItemSet& _subscribe：描述將要訂閱的屬性結構。

Return value：

None.

Exception：

- tvRTINameNotFound：指定作為象徵的名字在內容理不合法。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTIInteractionClassNotDefined：一個不合法的互動類別控制被當成服務引發的參數使用。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。

- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIFederateLoggingServiceCalls：這個例外不會被處理。
- tvRTIRegionNotKnown：一個 Region 物件不被確認為一個呼叫 getRegion 函數時所產生的合法 Region 物件。
- tvRTIInvalidRegionContext：企圖刪除一個仍與屬性實例有關聯的部位。

B.2 tvAttributeDeclare::~ tvAttributeDeclare

Description：

解除公開_publish 所描述的屬性結構以及解除訂閱_subscribe 所描述的結構。

Synopsis：

tvAttributeDeclare::

~tvAttributeDeclare();

Input value：

None.

Return value：

None.

Exception：

- tvRTIInteractionClassNotDefined：一個不合法的互動類別控制被當成服務引發的參數使用。
- tvRTIInteractionClassNotPublished：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 發佈的互動類別，但這個互動類別目前不是被這個 federate 發佈。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

- vRTIRegionNotKnown：一個 Region 物件不被確認為一個呼叫 getRegion 函數時所產生的合法 Region 物件。

B.3 tvAttributeDeclare:: Switch

Description：

告知有一個 RTI 事件被觸發了，此事件的訊息為_publish 所描述的屬性結構是否需要建立 RTI Object。

Synopsis：

```
virtual void
tvAttributeDeclare::
    Switch(
        tvRTIObjectClassHandle theClass,
        bool registration
    );
```

Input value：

- tvRTIObjectClassHandle theClass：此事件所指的物件類別 Handle。
- bool registration：此值如果為 true 表示 Federation 中至少有一個 Federate 訂閱了_publish 所描述的屬性，因務必要建立 RTI Object，如果為 false 則建議可以不需要建立 RTI Object。

Return value：

None.

Exception：

- tvRTIInteractionClassNotPublished：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 發佈的互動類別，但這個互動類別目前不是被這個 federate 發佈。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

C.1 tvDeclarationBase:: tvDeclarationBase

Description：

初始化關於 Declaration Management 所需的基本元件。

Synopsis：

```
tvDeclarationBase ::
    tvDeclarationBase(
        tvFedAgent * _agent
);
```

Input value：

- tvFedAgent * _agent：此 Federation 的 tvFedAgent 的物件指標。

Return value：

None.

Exception：

None.

D.1 tvDistributionBase :: tvDistributionBase

Description：

建立 RTI 的區域物件，以提供描述資料散佈範圍。

Synopsis：

```
tvDistributionBase ::
    tvDistributionBase(
        tvItemSet& regionDef
    );
```

Input value：

- tvItemSet& regionDef：描述區域的結構但是必須符合 FED 檔案內的空間定義。

Return value：

None.

Exception：

- tvRTISpaceNotDefined：一個不合法的物件控制被當作一個對 RTI 服務的參數。
- tvRTIInvalidExtents：實例範圍的寬度不合法。也就是說，下限高於上限或者限制長度在定義的[MIN_EXTENT,MAX_EXTENT]範圍之外。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTIRegionNotKnown：一個 Region 物件不被確認為一個呼叫 getRegion 函數時所產生的合法 Region 物件。

D.2 tvDistributionBase::~ tvDistributionBase

Description：

刪除由建構子建立的 RTI 區域物件。

Synopsis：

virtual

tvDistributionBase ::

~tvDistributionBase();

Input value：

None.

Return value：

None.

Exception：

- tvRTIRegionNotKnown：一個 Region 物件不被確認為一個呼叫 getRegion 函數時所產生的合法 Region 物件。
- tvRTIRegionInUse：
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

E.1 tvEntity:: tvEntity

Description：

建立 RTI 的 Object。

Synopsis：

```
tvEntity::
    tvEntity(
        tvFedAgent* _agent,
        const char* _name
    );
```

Input value：

- tvFedAgent* _agent：此 Federation 的 tvFedAgent 的物件指標。
- const char* _name：RTI 的 Object 名稱。

Return value：

None.

Exception：

- tvRTIObjectClassNotDefined：一個不合法的物件案例控制被使用。
- tvRTIObjectClassNotPublished：一個服務或者回呼引發期待他的主題是一個目前被 local federate 發佈的物件類別，但這類別不是被 local federate 發佈。
- tvRTIObjectAlreadyRegistered：企圖對 federation 以一個非唯一的象徵性名字註冊。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。

- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgres：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTIAttributeNotDefined：一個屬性控制被不合法使用在特定的的物件類別或者物件實例的內容。
- tvRTIAttributeNotPublished：一個服務或者回呼被引發期待他的主題是一個 local federate 目前所發佈的類別屬性，但這個 federate 不是目前發佈的屬性。
- tvRTIRegionNotKnown：一個 Region 物件不被確認為一個呼叫 getRegion 函數時所產生的合法 Region 物件。
- tvRTIInvalidRegionContext：企圖刪除一個仍與屬性實例有關聯的部位。

E.2 tvEntity:: ~tvEntity

Description：

刪除由建構子建立的 RTI Object。

Synopsis：

virtual

tvEntity::

~tvEntity();

Input value：

None.

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIDeletePrivilegeNotHeld：企圖刪除一個沒擁有 privilegeToDelete 的屬性實例的 local federate 中的一個物件。
- tvRTIInvalidFederationTime：一個對回呼或者服務的邏輯時間參數不是一個 federation 邏輯時間軸合法的點。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTIFederateInternalErro：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

E.3 tvEntity:: Rediscover

Description：

暫時的刪除 RTI Object 使得其他的 Federate 可以重新發現此 RTI Object。

Synopsis：

void

tvEntity::

Rediscover();

Input value：

None.

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIFederateOwnsAttributes：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 所擁有的屬性實例，但 local federate 目前沒有擁有這個屬性實例。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

E.4 tvEntity:: RequestValue

Description：

要求其他 Federate 更新特定的屬性狀態。

Synopsis：

```
void
tvEntity::
    RequestValue(
        tvRTIAttributeHandleSet& theAttributes
    );
```

Input value：

- tvRTIAttributeHandleSet& theAttributes：指定的屬性 Handle 集合。

Return value：

None.

Exception：

- tvRTIObjectClassNotDefined：一個不合法的物件案例控制被使用。
- tvRTIAttributeNotDefined：一個屬性控制被不合法使用在特定的的物件類別或者物件實例的內容。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIRegionNotKnown：一個 Region 物件不被確認為一個呼叫 getRegion 函數時所產生的合法 Region 物件。

E.5&E.6 tvEntity:: Send

Description：

更新指定的屬性的狀態值。

Synopsis：

```
void
tvEntity::
    Send(
        tvRTIAttributeHandleValuePairSet* _set,
        tvRTIFedTime theTime
    );

void
tvEntity::
    Send(
        tvRTIAttributeHandleValuePairSet* _set
    );
```

Input value：

- tvRTIAttributeHandleValuePairSet* _set：欲更新的屬性狀態值集合。
- tvRTIFedTime theTime：指定執行此動作的模擬時間。

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIAttributeNotDefined：一個屬性控制被不合法使用在特定的的物件類別或者物件實例的內容。
- tvRTIAttributeNotOwned：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 擁有的屬性實例，但是 local federate 目前沒有擁有這個屬性實例。

- tvRTIInvalidFederationTime：一個對回呼或者服務的邏輯時間參數不是一個 federation 邏輯時間軸合法的點。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

E.7 tvEntity:: Transport

Description：

設定屬性狀態傳輸的模式。

Synopsis：

```
void
tvEntity::
    Transport(
        tvRTIAttributeHandleSet& _theAttributes,
        tvRTITransportationHandle _theType
    );
```

Input value：

- tvRTIAttributeHandleSet& _theAttributes：欲更改傳輸模式的屬性 Handle 集合。
- tvRTITransportationHandle _theType：傳輸模式的 Handle。

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIAttributeNotDefined：一個屬性控制被不合法使用在特定的的物件類別或者物件實例的內容。
- tvRTIAttributeNotOwned：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 擁有的屬性實例，但是 local federate 目前沒有擁有這個屬性實例。
- tvRTIInvalidTransportationHandle：一個安排的控制不被確認為一個由 getTransportationHandle()回傳的合法控制。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI

ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。

- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTIAttributeNotOwned：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 擁有的屬性實例，但是 local federate 目前沒有擁有這個屬性實例。

E.8 tvEntity:: Receive

Description：

接收由 Federate 更新的物件屬性狀態值。

Synopsis：

```
virtual void
tvEntity::
    Receive(
        tvRTIAttributeHandleValuePairSet& _set
    );
```

Input value：

- tvRTIAttributeHandleValuePairSet& _set：物件屬性狀態值得集合。

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIAttributeNotKnown：一個屬性控制被使用在不合法的指定類別或者物件實例。
- tvRTIFederateOwnsAttributes：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 所擁有的屬性實例，但 local federate 目前沒有擁有這個屬性實例。
- tvRTIInvalidFederationTime：一個對回呼或者服務的邏輯時間參數不是一個 federation 邏輯時間軸合法的
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

E.9 tvEntity:: Refresh

Description：

接收由其他 Federate 提出的更新要求，並立即的更新指定的屬性狀態值。

Synopsis：

```
virtual void
tvEntity::
    Refresh(
        tvRTIAttributeHandleSet& _set
    );
```

Input value：

None.

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIAttributeNotKnown：一個屬性控制被使用在不合法的指定類別或者物件實例。
- tvRTIFederateOwnsAttributes：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 所擁有的屬性實例，但 local federate 目前沒有擁有這個屬性實例。
- tvRTIInvalidFederationTime：一個對回呼或者服務的邏輯時間參數不是一個 federation 邏輯時間軸合法的
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

E.10 tvEntity:: ScopeAlert

Description：

當有需要接收或傳送的屬性實體進入目前的訂閱區域(Subscribe region)或更新區域(Update region)

Synopsis：

```
void
tvEntity::
    ScopeAlert(
        bool scope,
        tvRTIObjectHandle _handle,
        tvRTIAttributeHandleSet& _set
    );
```

Input value：

- bool scope：當此事件為屬性實體進入時此值為 true，如果為屬性實體離開時此值為 false。
- tvRTIObjectHandle _handle：這些屬性集合所屬的 RTI Object Handle。
- tvRTIAttributeHandleSet& _set：符合此事件的屬性 Handle 集合

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIAttributeNotKnown：一個屬性控制被使用在不合法的指定類別或者物件實例。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

E.11 tvEntity:: Switch

Description：

當有其他的 Federate 發現這個 RTI Object，則會引發此事件。

Synopsis：

```
virtual void
tvEntity::
    Switch(
        tvRTIAttributeHandleSet& theAttributes,
        bool _switch
    );
```

Input value：

- tvRTIAttributeHandleSet& theAttributes：需要更新的屬性 Handle 集合。
- bool _switch：此值為 true 時代表上述的屬性比需要做狀態更新的動作，反之則可以不需要做狀態更新的動作。

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIAttributeNotOwned：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 擁有的屬性實例，但是 local federate 目前沒有擁有這個屬性實例。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

G.1 tvFederationBase:: tvFederationBase

Description：

建立 Federation Execution 並加入此 Federation。

Synopsis：

```
tvFederationBase::
    tvFederationBase(
        const char* _executionName,
        const char* _federateName ,
        const char* _fed
    );
```

Input value：

- const char* _executionName：Federation Execution 的名稱。
- const char* _federateName ：Federate 的名稱。
- const char* _fed：FED 檔案路徑。

Return value：

None.

Exception：

- tvRTIFederationExecutionAlreadyExists：企圖去創造一個已存在的 federation execution。
- tvRTICouldNotOpenFED：使用者所提供的 FED 檔案在指定的位置無法尋得。
- tvRTIErrorReadingFED：在指定的檔案位置找到 FED 檔案，但不是正確的格式。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

- tvRTIFederateAlreadyExecutionMember：企圖在 RTI ambassdor 已經加入一個 federation 後加入一個 federation execution。
- tvRTIFederationExecutionDoesNotExist：企圖去加入一個仍未存在的 federation execution。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。

G.2 tvFederationBase::~ tvFederationBase

Description：

離開 Federation Execution 並嘗試刪除 Federation Execution。

Synopsis：

virtual

tvFederationBase::

~tvFederationBase();

Input value：

None.

Return value：

None.

Exception：

- tvRTIFederateOwnsAttributes：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 所擁有的屬性實例，但 local federate 目前沒有擁有這個屬性實例。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIInvalidResignAction：resign-action 參數不被確認為一個合法的 ResignAction 型態。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTIFederatesCurrentlyJoined：企圖刪除一個還有 federate 加入的 federation execution。
- tvRTIFederationExecutionDoesNotExist：企圖去加入一個仍未存在的 federation execution。

G.3 tvFederationBase:: GetExecutionName

Description：

取得 Federation Execution 的名稱。

Synopsis：

const char*

tvFederationBase::

GetExecutionName();

Input value：

None.

Return value：

Federation Execution 名稱的字串指標。

Exception：

None.

G.4 tvFederationBase:: GetFederateHandle

Description：

取的 Federate 的 Handle。

Synopsis：

tvRTIFederateHandle

tvFederationBase::

GetFederateHandle();

Input value：

None.

Return value：

Federate Handle 值。

Exception：

None.

G.5 tvFederationBase:: GetFederateName

Description：

取得 Federate 名稱。

Synopsis：

const char*

tvFederationBase::

GetFederateName();

Input value：

None.

Return value：

Federate 名稱的字串指標。

Exception：

None.

G.6 tvFederationBase:: GetRestoreLabel

Description：

取得 Restore 標籤的字串。

Synopsis：

const char*

tvFederationBase::

GetRestoreLabel();

Input value：

None.

Return value：

Restore 標籤的字串指標。

Exception：

None.

G.7 tvFederationBase::GetSaveLabel

Description：

取得 Save 標籤的字串指標。

Synopsis：

const char*

tvFederationBase::

GetSaveLabel();

Input value：

None.

Return value：

Save 標籤的字串指標。

Exception：

None.

G.8 tvFederationBase:: IsFedExecCreator

Description：

判斷自己是否為 Federation Execution 的建立者。

Synopsis：

bool

tvFederationBase::

IsFedExecCreator();

Input value：

None.

Return value：

如果 Federation Execution 是自己建立的則傳回 true，反之傳回 false。

Exception：

None.

G.9 tvFederationBase:: Restore

Description：

回復 Federation 的內部狀態值。

Synopsis：

```
void
tvFederationBase::
    Restore(
        const char* label
    );
```

Input value：

- const char* label：回覆動作的辨識標籤。

Return value：

None.

Exception：

- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。
- tvRTISpecifiedSaveLabelDoesNotExist：這個 federate 被要求恢復一個不符合一個儲存的儲存標記。
- tvRTICouldNotRestore：這個例外應該不被 federate 丟出;反而使用 federateRestoreNotComplete()。
- tvRTIRestoreNotRequested：當一個 federate 沒有被要求復原，但是 federate 回報一個完整的恢復企圖。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。

- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

G.10 tvFederationBase:: Save

Description：

儲存 Federation 的內部狀態值。

Synopsis：

```
void
tvFederationBase::
    Save(
        const char* label
    );
```

Input value：

- const char* label：儲存動作的辨識標籤。

Return value：

None.

Exception：

- tvRTIFederationTimeAlreadyPassed：企圖將存檔排程或者推進 federate 的邏輯時間到 federation 已經過的邏輯時間。
- tvRTIInvalidFederationTime：一個對回呼或者服務的邏輯時間參數不是一個 federation 邏輯時間軸合法的點。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

- tvRTISaveNotInitiated：當一個 federate 沒有被要求儲存，但卻回報一個完整的儲存企圖。
- tvRTIUnableToPerformSave：這個例外應該不被這個 federate 丟出去;而使用 federateSaveNotComplete()。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

G.11&G.12 tvFederationBase:: Tick

Description：

驅動 RTI 的指令，當程式有呼叫 RTI 的 API 或動作時，隨後必須呼叫此函式驅動並執行。

Synopsis：

```
void
tvFederationBase::
    Tick();

bool
tvFederationBase::
    Tick(
        double min,
        double max
    );
```

Input value：

- double min：此函式工作時間的下限。
- double max：此函式工作時間的上限。

Return value：

如果傳回 true 代表 RTI 內部尚有未完成的工作，反之則傳回 false。

Exception：

- tvRTISpecifiedSaveLabelDoesNotExist：這個 federate 被要求恢復一個不符合一個儲存的儲存標記。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

H.1 tvFedExec:: tvFedExec

Description：

初始化 tvFederationBase 物件。

Synopsis：

```
tvFedExec::
    tvFedExec(
        const char* _executionName,
        const char* _federateName,
        const char* _fed
    );
```

Input value：

- const char* _executionName：Federation Execution 的名稱。
- const char* _federateName ：Federate 的名稱。
- const char* _fed：FED 檔案路徑。

Return value：

None.

Exception：

None.

H.2 tvFedExec:: Syn_Achieve

Description：

通知 RTI 此 Federate 已經到達最近一次的 Synchronization Point。

Synopsis：

void

tvFedExec::

Syn_Achieve();

Input value：

None.

Return value：

None.

Exception：

- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。
- tvRTISynchronizationPointLabelWasNotAnnounced：當沒有重要的同步要求時，federate 回報達到同步點。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

H.3 tvFedExec:: Syn_Registration

Description：

啟動 Federate Synchronization 的功能，呼叫此函式之後會一直等到此 Federation 的所有 Federate 皆到達了之後才回 return。

Synopsis：

bool

tvFedExec::

Syn_Registration()

Input value：

None.

Return value：

None.

Exception：

- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

I.1 tvFedTime:: Query

Description：

Synopsis：

tvRTIFedTime

tvFedTime::

Query(

int _queryKind

);

Input value：

- int _queryKind：欲查詢邏輯時間的種類，可以輸入以下三種 TV_QUERY_FEDERATE_TIME、TV_QUERY_LBTS 以及 TV_QUERY_MIN_NEXT_EVENT_TIME。

Return value：

None.

Exception：

- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

J.1&J.2 tvInteraction:: Send

Description：

送出 Interaction。

Synopsis：

```
void
tvInteraction::
    Send(
        tvRTIParameterHandleValuePairSet& theParameters
    );

void
tvInteraction::
    Send(
        tvRTIParameterHandleValuePairSet& _theParameters ,
        tvRTIFedTime _theTime
    );
```

Input value：

- tvRTIParameterHandleValuePairSet& _theParameters ： 此 Interaction 所包含的參數(Parameter)內容集合。
- tvRTIFedTime _theTime：此 Interaction 的執行邏輯時間。

Return value：

None.

Exception：

- tvRTIInteractionClassNotDefined：一個不合法的互動類別控制被當成服務引發的參數使用。
- tvRTIInteractionClassNotPublished：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 發佈的互動類別，但這個互動類別目前不是被這個 federate 發佈。
- tvRTIInteractionParameterNotDefined：一個參數控制被不合法使用在一個互動類別內容裡。

- tvRTIInvalidFederationTime：
- 一個對回呼或者服務的邏輯時間參數不是一個 federation 邏輯時間軸合法的點。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTIRegionNotKnown：一個 Region 物件不被確認為一個呼叫 getRegion 函數時所產生的合法 Region 物件。
- tvRTIInvalidRegionContext：企圖刪除一個仍與屬性實例有關聯的部位。

J.3 tvInteraction:: Transport

Description：

設定 Interaction 的網路傳輸模式。

Synopsis：

```
void
tvInteraction::
    Transport(
        tvRTITransportationHandle _theType
    );
```

Input value：

tvRTITransportationHandle _theType：傳輸模式的 Handle。

Return value：

None.

Exception：

- tvRTIInteractionClassNotDefined：一個不合法的互動類別控制被當成服務引發的參數使用。
- tvRTIInteractionClassNotPublished：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 發佈的互動類別，但這個互動類別目前不是被這個 federate 發佈。
- tvRTIInvalidTransportationHandle：一個安排的控制不被確認為一個由 getTransportationHandle()回傳的合法控制。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。

- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

J.4 tvInteraction:: Receive

Description：

接收來自其他 Federate 發出的 Interaction 訊息。

Synopsis：

```
virtual void
tvInteraction::
    Receive(
        tvRTIParameterHandleValuePairSet& _theParameters
    );
```

Input value：

- tvRTIParameterHandleValuePairSet& _theParameters：Interaction 的參數值集合。

Return value：

None.

Exception：

- tvRTIInteractionClassNotKnown：一個不合法的互動類別控制被當呈給服務引發的參數使用。
- tvRTIInteractionParameterNotKnown：一個參數控制背部合法的指定互動類別內容使用。
- tvRTIInvalidFederationTime：一個對回呼或者服務的邏輯時間參數不是一個 federation 邏輯時間軸合法的點。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

K.1 tvInteractionDDM:: tvInteractionDDM

Description：

初始化 tvInteractionDeclare 與 tvDistributionBase 的實體，並且使 Interaction 與特定的資料散佈範圍的區域產生關聯。

Synopsis：

```
tvInteractionDDM::
    tvInteractionDDM(
        tvFedAgent* _agent,
        tvItemSet& _publish ,
        tvItemSet& _subscribe,
        tvItemSet& regionDef ,
        bool DDM_Region=false
    );
```

Input value：

- tvFedAgent* _agent：此 Federation 的 tvFedAgent 的物件指標。
- tvItemSet& _publish ：描述將要公開的 Interaction 結構。
- tvItemSet& _subscribe：描述將要訂閱的 Interaction 結構。
- tvItemSet& regionDef ：描述區域的結構但是必須符合 FED 檔案內的空間定義。
- bool DDM_Region：此參數預設值為 false，代表不限定 Interaction 的範圍，如果輸入 true 值，則 Interaction 才會與資料散佈範圍的區域產生關聯。

Return value：

None.

Exception：

L.1 tvInteractionDeclare::tvInteractionDeclare

Description：

Synopsis：

```
tvInteractionDeclare::
    tvInteractionDeclare(
        tvFedAgent* _agent ,
        tvItemSet& _publish ,
        tvItemSet& _subscribe
    );
```

Input value：

- tvFedAgent* _agent ：此 Federation 的 tvFedAgent 的物件指標。
- tvItemSet& _publish ：描述將要公開的 Interaction 結構。
- tvItemSet& _subscribe：描述將要訂閱的 Interaction 結構。

Return value：

None.

Exception：

- tvRTINameNotFound：指定作為象徵的名字在內容理不合法。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTIInteractionClassNotDefined：一個不合法的互動類別控制被當成服務引發的參數使用。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。

- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIFederateLoggingServiceCalls：這個例外不會被處理。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRegionNotKnown：一個 Region 物件不被確認為一個呼叫 getRegion 函數時所產生的合法 Region 物件。
- tvRTIInvalidRegionContext：企圖刪除一個仍與屬性實例有關聯的部位。

L.2 tvInteractionDeclare::~tvInteractionDeclare

Description：

解除公開_publish 所描述的 Interaction 結構以及解除訂閱_subscribe 所描述的 Interaction 結構。

Synopsis：

virtual

tvInteractionDeclare::

~tvInteractionDeclare();

Input value：

None.

Return value：

None.

Exception：

- tvRTIInteractionClassNotDefined：一個不合法的互動類別控制被當成服務引發的參數使用。
- tvRTIInteractionClassNotPublished：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 發佈的互動類別，但這個互動類別目前不是被這個 federate 發佈。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

- tvRTIInteractionClassNotSubscribed：一個服務或者回呼引發期待他的主題是一個不是目前被 local federate 訂閱的互動類別，但是這個互動類別目前被 federate 訂閱。

L.3 tvInteractionDeclare::Switch

Description：

告知有一個 RTI 事件被觸發了，此事件的訊息為_publish 所描述的 Interaction 結構是否需要在執行時期送出 Interaction。

Synopsis：

```
virtual void
tvInteractionDeclare::
    Switch(
        tvRTIInteractionClassHandle theHandle,
        bool interactionSwitch
    );
```

Input value：

- tvRTIInteractionClassHandle theHandle：此事件所指的 Interaction 類別 Handle。
- bool interactionSwitch：此值如果為 true 表示 Federation 中至少有一個 Federate 訂閱了_publish 所描述的 Interaction，故務必要送出此類別的 Interaction，如果為 false 則建議可以不需要送出。

Return value：

None.

Exception：

- tvRTIInteractionClassNotPublished：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 發佈的互動類別，但這個互動類別目前不是被這個 federate 發佈。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

M.1 tvInteractionTime: ChangeOrder

Description：

設定這個 Federate 有關發出 Interaction 的排列順序方法，可以是按照時間標記或按照收到封包的先後。

Synopsis：

```
void
tvInteractionTime::
    ChangeOrder(
        tvRTIInteractionClassHandle _theClass,
        tvRTIOrderingHandle _theType
    );
```

Input value：

- tvRTIInteractionClassHandle _theClass：欲設定的 Interaction 類別 Handle。
- tvRTIOrderingHandle _theType：排列順序方法的 Handle。

Return value：

None.

Exception：

None.

N.1 tvItemSet::tvItemSet

Description：

儲存這些元素的類別名稱。

Synopsis：

```
tvItemSet::
    tvItemSet(
        const char* _className
    );
```

Input value：

- const char* _className：類別名稱。

Return value：

None.

Exception：

None.

N.2 tvItemSet:: Add

Description：

新增一個元素的名稱。

Synopsis：

```
bool
tvItemSet::
    Add(
        const char* _name
    );
```

Input value：

- const char* _name：元素名稱。

Return value：

成功傳回 true，如果已經存在了就回傳回 false。

Exception：

None.

N.3 tvItemSet:: GetClassName

Description：

取得類別名稱。

Synopsis：

string

tvItemSet::

GetClassName();

Input value：

None.

Return value：

類別名稱的字串物件。

Exception：

None.

N.4 tvItemSet:: GetItemName

Description：

取得元素的名稱。

Synopsis：

```
string
tvItemSet::
    GetItemName(
        int _index
    );
```

Input value：

- int _index：元素索引值，第一個元素的索引值為 0。

Return value：

元素名稱的字串物件。

Exception：

None.

N.5 tvItemSet:: Size

Description：

取得元素個數。

Synopsis：

int

tvItemSet::

Size();

Input value：

None.

Return value：

元素個數值。

Exception：

None.

O.1 tvObjectRoot::Advisory

Description：

設定 RTI 提示機制的開啟狀態，包括是否需要進行狀態更新動作的提示、是否需要送出 Interaction 動作的提示、是否需要建立 RTI Object 的提示以及是否有屬性實體進入訂閱或更新區域的提示。

Synopsis：

```
void
tvObjectRoot::
    Advisory(
        int _switchKind,
        bool _switch
    );
```

Input value：

- nt _switchKind：欲啟動的種類，可以是 TV_ATTRIBUTE_UPDATE , TV_ATTRIBUTE_SCOPE , TV_CLASS_UPDATE , TV_INTERACTION_RELEVANCE 的其中一種。
- bool _switch：此值為 true 代表啟動，反之為關閉功能。

Return value：

None.

Exception：

- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。

- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

P.1 tvObjectTime::ChangeOrder

Description：

設定這個 Federate 有關發出物件屬性的排列順序方法，可以是按照時間標記或按照收到封包的先後。

Synopsis：

```
void
tvObjectTime::
    ChangeOrder(
        tvRTIObjectHandle _theObject,
        tvRTIAttributeHandleSet& _theAttributes,
        tvRTIOrderingHandle _theType
    );
```

Input value：

- tvRTIObjectHandle _theObject：欲設定的屬性物件類別的 Handle。
- tvRTIAttributeHandleSet& _theAttributes：欲設定的屬性 Handle 的集合。
- tvRTIOrderingHandle _theType：排列順序方法的 Handle。

Return value：

None.

Exception：

None.

Q.1 tvOwnershipBase:: Abandon

Description：

設定特定的屬性的控制權是可以被轉移的。

Synopsis：

```
void
tvOwnershipBase::
    Abandon(
        tvItemSet& _attrSet
    );
```

Input value：

- tvItemSet& _attrSet：欲設定的屬性集合。

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIAttributeNotDefined：一個屬性控制被不合法使用在特定的的物件類別或者物件實例的內容。
- tvRTIAttributeNotOwned：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 擁有的屬性實例，但是 local federate 目前沒有擁有這個屬性實例。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。

- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

Q.2 tvOwnershipBase:: Discard

Description：

主動提出讓出特定屬性的控制權。

Synopsis：

```
void
tvOwnershipBase::
    Discard(
        tvItemSet &_attrSet
    );
```

Input value：

- tvItemSet& _attrSet：欲設定的屬性集合。

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIAttributeNotDefined：一個屬性控制被不合法使用在特定的的物件類別或者物件實例的內容。
- tvRTIAttributeNotOwned：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 擁有的屬性實例，但是 local federate 目前沒有擁有這個屬性實例。
- tvRTIAttributeAlreadyBeingDivested：要求已經有一個重要除去需求的 local federate 去除掉一個 attribute-instance。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。

- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIAttributeNotKnown：一個屬性控制被使用在不合法的指定類別或者物件實例。
- tvRTIAttributeDivestitureWasNotRequested：一個服務或者回呼被引發期待他的主題是一個重要的除去要求屬性實例，但是 local federate 沒有重要的除去要求。

Q.3 tvOwnershipBase:: Request

Description：

主動要求特定屬性的控制權。

Synopsis：

```
void
tvOwnershipBase::
    Request(
        tvItemSet &_attrSet
    );
```

Input value：

- tvItemSet& _attrSet：欲設定的屬性集合。

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIObjectClassNotPublished：一個服務或者回呼引發期待他的主題是一個目前被 local federate 發佈的物件類別，但這類別不是被 local federate 發佈。
- tvRTIAttributeNotDefined：一個屬性控制被不合法使用在特定的的物件類別或者物件實例的內容。
- tvRTIAttributeNotPublished：一個服務或者回呼被引發期待他的主題是一個 local federate 目前所發佈的類別屬性，但這個 federate 不是目前發佈的屬性。
- tvRTIFederateOwnsAttributes：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 所擁有的屬性實例，但 local federate 目前沒有擁有這個屬性實例。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。

- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTIAttributeNotKnown：一個屬性控制被使用在不合法的指定類別或者物件實例。
- tvRTIAttributeNotOwned：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 擁有的屬性實例，但是 local federate 目前沒有擁有這個屬性實例。
- tvRTIAttributeDivestitureWasNotRequested：一個服務或者回呼被引發期待他的主題是一個重要的除去要求屬性實例，但是 local federate 沒有重要的除去要求。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。
- tvRTIAttributeAlreadyOwned：一個服務或回呼被引發期待他的主題是一個非同時被 local federate 擁有的屬性實例，但是 local federate 卻是同時擁有這個案件屬性。
- tvRTIAttributeAcquisitionWasNotCanceled：一個屬性取得的確定刪除回呼被產生給不是先前取得刪除主題的屬性實例。

Q.4 tvOwnershipBase:: QueryLocal

Description：

查詢特定的屬性是否為自己(Federate)所控制的。

Synopsis：

```
bool
tvOwnershipBase::
    QueryLocal(
        tvItemSet &_attrSet
    );
```

Input value：

- tvItemSet& _attrSet：欲查詢的屬性集合。

Return value：

如果為自己所控制的傳回 true，反之傳回 false。

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIAttributeNotDefined：一個屬性控制被不合法使用在特定的的物件類別或者物件實例的內容。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- **tvRTIRTIinternalError**：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

Q.5 tvOwnershipBase:: QueryOwner

Description：

查詢特定屬性的控制權擁有者。

Synopsis：

```
int
tvOwnershipBase::
    QueryOwner(
        tvRTIAttributeHandle _handle
    );
```

Input value：

- tvRTIAttributeHandle _handle：欲查詢的屬性 Handle。

Return value：

擁有者代碼。

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIAttributeNotDefined：一個屬性控制被不合法使用在特定的的物件類別或者物件實例的內容。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。
- tvRTIAttributeNotKnown：一個屬性控制被使用在不合法的指定類別或者物件實例。

Q.6 tvOwnershipBase:: Unobtrusive

Description：

主動取得屬性的控制權，但不一定成功。

Synopsis：

```
void
tvOwnershipBase::
    Unobtrusive(
        tvItemSet &_attrSet
    );
```

Input value：

- tvItemSet& _attrSet：欲取得控制權的屬性集合。

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIObjectClassNotPublished：一個服務或者回呼引發期待他的主題是一個目前被 local federate 發佈的物件類別，但這類別不是被 local federate 發佈。
- tvRTIAttributeNotDefined：一個屬性控制被不合法使用在特定的的物件類別或者物件實例的內容。
- tvRTIAttributeNotPublished：一個服務或者回呼被引發期待他的主題是一個 local federate 目前所發佈的類別屬性，但這個 federate 不是目前發佈的屬性。
- tvRTIFederateOwnsAttributes：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 所擁有的屬性實例，但 local federate 目前沒有擁有這個屬性實例。
- tvRTIAttributeAlreadyBeingAcquired：要求已經有一個重要取得需求的 local federate 去取得一個 attribute-instance。

- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTIAttributeNotKnown：一個屬性控制被使用在不合法的指定類別或者物件實例。
- tvRTIAttributeAlreadyOwned：一個服務或回呼被引發期待他的主題是一個非同時被 local federate 擁有的屬性實例，但是 local federate 卻是同時擁有這個案件屬性。
- tvRTIAttributeAcquisitionWasNotRequested：一個服務或回呼被引發期待他的主題是一個重要的取得要求，但是 local federate 沒有重要的取得要求。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

Q.7 tvOwnershipBase:: Accept

Description：

通知程式已經接受其他 Federate 以 Push 方式釋放出來的屬性控制權。

Synopsis：

```
virtual void
tvOwnershipBase::
    Accept(
        tvRTIAttributeHandleSet *_theAttributes
    );
```

Input value：

- tvRTIAttributeHandleSet *_theAttributes：接受控制權的屬性集合。

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIAttributeNotKnown：一個屬性控制被使用在不合法的指定類別或者物件實例。
- tvRTIAttributeAlreadyOwned：一個服務或回呼被引發期待他的主題是一個非同時被 local federate 擁有的屬性實例，但是 local federate 卻是同時擁有這個案件屬性。
- tvRTIAttributeNotPublished：一個服務或者回呼被引發期待他的主題是一個 local federate 目前所發佈的類別屬性，但這個 federate 不是目前發佈的屬性。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。
- tvRTIAttributeAcquisitionWasNotRequested：一個服務或回呼被引發期待他的主題是一個重要的取得要求，但是 local federate 沒有重要的取得要求。

- tvRTIObjectClassNotPublished：一個服務或者回呼引發期待他的主題是一個目前被 local federate 發佈的物件類別，但這類別不是被 local federate 發佈。
- tvRTIAttributeNotDefined：一個屬性控制被不合法使用在特定的的物件類別或者物件實例的內容。
- tvRTIAttributeNotPublished：一個服務或者回呼被引發期待他的主題是一個 local federate 目前所發佈的類別屬性，但這個 federate 不是目前發佈的屬性。
- tvRTIFederateOwnsAttributes：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 所擁有的屬性實例，但 local federate 目前沒有擁有這個屬性實例。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

Q.8 tvOwnershipBase:: Relinquish

Description：

通知程式已經接受其他 Federate 以 Pull 方式釋放出來的屬性控制權。

Synopsis：

```
virtual void
tvOwnershipBase::
    Relinquish(
        tvRTIAttributeHandleSet* _theAttributes
    );
```

Input value：

- tvRTIAttributeHandleSet *_theAttributes：接受控制權的屬性集合。

Return value：

None.

Exception：

- tvRTIObjectNotKnown：一個物件實例控制對服務或者回呼的參數個別地不符合 LRC 或這個 federate 所知到的物件實例。
- tvRTIAttributeNotDefined：一個屬性控制被不合法使用在特定的的物件類別或者物件實例的內容。
- tvRTIAttributeNotOwned：一個服務或者回呼被引發期待他的主題是一個目前被 local federate 擁有的屬性實例，但是 local federate 目前沒有擁有這個屬性實例。
- tvRTIFederateWasNotAskedToReleaseAttribute：當沒有其他要求比這個屬性實例更加重要的時候，federate 嘗試回應一個屬性實例釋放要求。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。

- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIAttributeNotKnown：一個屬性控制被使用在不合法的指定類別或者物件實例。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

R.1&R.2&R.3 tvTimeBase:: tvTimeBase

Description：

設定 Federate 的時間進行模式，第一種建構子可建構出具有 Time Constrained 時間行進模式模式的 Federate，第二種建構子可建構出具有 Time Constrained 與 Time Regulating 時間行進模式模式或具有 Time Regulating 時間行進模式模式的 Federate，第三種則是不使用時間管理機制。

Synopsis：

```
tvTimeBase::
    tvTimeBase(
        bool constrain
    );

tvTimeBase::
    tvTimeBase(
        bool constrain,
        tvRTIFedTime& theLookahead
    );

tvTimeBase::
    tvTimeBase();
```

Input value：

- bool constrain：如果設為 true 代表具有 Time Constrained 的時間行進模式，反之則代表不使用時間管理機制。
- tvRTIFedTime& theLookahead：設定 Time Regulating 的 Lookahead 值。

Return value：

None.

Exception：

- tvRTITimeRegulationAlreadyEnabled：當時間管理已經啟動，federate 企圖啟動時間管理。

- tvRTIEnableTimeRegulationPending：企圖在一個要求開啟的時間管理正在進行中，開啟時間管理或者推進時間。
- tvRTITimeAdvanceAlreadyInProgress：在先前的時間進行服務仍未結束前引發另一個時間前進服務。
- tvRTIInvalidFederationTime：一個對回呼或者服務的邏輯時間參數不是一個 federation 邏輯時間軸合法的點。
- tvRTIInvalidLookahead：企圖開啟管理或者以一個不合法的邏輯時間參數修改 federate。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTITimeRegulationWasNotEnabled：當時間管理已經關閉，federate 企圖關閉時間管理。
- tvRTIEnableTimeRegulationWasNotPending：federate 接到一個回呼告知時間管理已經被啟動，但他沒有要求時間管理啟動。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。
- tvRTITimeConstrainedAlreadyEnabled：當時間限制已經啟動，federate 企圖啟動時間限制。
- tvRTIEnableTimeConstrainedPending：企圖在一個要求開啟的時間限制正在進行中，開啟時間限制或者推進時間。

- tvRTITimeConstrainedWasNotEnabled：當時間限制已經關閉，federate 企圖關閉時間限制服務。
- tvRTIEnableTimeConstrainedWasNotPending：federate 接到一個回呼告知時間限制已經被啟動，但他沒有要求時間限制啟動。

R.4 tvTimeBase:: Delivery

Description：

設定是否使用非同步傳送。

Synopsis：

```
void
tvTimeBase::
    Delivery(
        bool _asyn
    );
```

Input value：

- bool _asyn：如果設為 true 代表開啟非同步傳送的功能，反之代表關閉此功能。

Return value：

None.

Exception：

- tvRTIAsynchronousDeliveryAlreadyEnabled：啟動在非同步傳遞已經啟動時被產生的有接收順序的事件的非同步傳遞。
- tvRTIAsynchronousDeliveryAlreadyDisabled：關閉在非同步傳遞已經關閉時被產生的有接收順序的事件的非同步傳遞。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。

- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

R.5 tvTimeBase:: EventTime

Description：

以事件驅動的方式前進到下一個時間。

Synopsis：

tvRTIFedTime
tvTimeBase::
EventTime(
bool _available=false
);

Input value：

bool _available：設定是否要立即的前進到下一個時間。

Return value：

目前的模擬時間。

Exception：

- tvRTIInvalidFederationTime：一個對回呼或者服務的邏輯時間參數不是一個 federation 邏輯時間軸合法的點。
- tvRTIFederationTimeAlreadyPassed：企圖將存檔排程或者推進 federate 的邏輯時間到 federation 已經過的邏輯時間。
- tvRTITimeAdvanceAlreadyInProgress：在先前的時間進行服務仍未結束前引發另一個時間前進服務。
- tvRTIEnableTimeRegulationPending：企圖在一個要求開啟的時間管理正在進行中，開啟時間管理或者推進時間。
- tvRTIEnableTimeConstrainedPending：企圖在一個要求開啟的時間限制正在進行中，開啟時間限制或者推進時間。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。

- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTIEnableTimeConstrainedPending：企圖在一個要求開啟的時間限制正在進行中，開啟時間限制或者推進時間。
- tvRTITimeAdvanceWasNotInProgress：當沒有任何時間前進服務進行時，federate 被告知一個時間前進的允許。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

R.6&R.7 tvTimeBase:: Lookahead

Description：

查詢或修改 Lookahead。

Synopsis：

```
tvRTIFedTime
    tvTimeBase::
        Lookahead();

void
tvTimeBase::
    Lookahead(
        tvRTIFedTime _modifyTime
    );
```

Input value：

- tvRTIFedTime _modifyTime：欲修改的 Lookahead 值。

Return value：

查詢到的 Lookahead 值。

Exception：

- tvRTIInvalidLookahead：企圖開啟管理或者以一個不合法的邏輯時間參數修改 federate。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。

- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

R.8 tvTimeBase:: Optimistic

Description：

前進到下一個時間點，但是此函式會先確定所有訊息皆已經到達了才會return。

Synopsis：

tvRTIFedTime

tvTimeBase::

Optimistic();

Input value：

None.

Return value：

目前的模擬時間。

Exception：

- tvRTIInvalidFederationTime：一個對回呼或者服務的邏輯時間參數不是一個 federation 邏輯時間軸合法的點。
- tvRTIFederationTimeAlreadyPassed：企圖將存檔排程或者推進 federate 的邏輯時間到 federation 已經過的邏輯時間。
- tvRTITimeAdvanceAlreadyInProgress：在先前的時間進行服務仍未結束前引發另一個時間前進服務。
- tvRTIEnableTimeConstrainedPending：企圖在一個要求開啟的時間限制正在進行中，開啟時間限制或者推進時間。
- tvRTIEnableTimeRegulationPending：企圖在一個要求開啟的時間管理正在進行中，開啟時間管理或者推進時間。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。

- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTITimeAdvanceWasNotInProgress：當沒有任何時間前進服務進行時，federate 被告知一個時間前進的允許。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

R.9 tvTimeBase:: Retract

Description：

取消先前的含有時間標記且目前模擬時間還沒到達的 RTI 命令或更新動作。

Synopsis：

```
void
tvTimeBase::
    Retract(
        tvRTIEventRetractionHandle _rh
    );
```

Input value：

- tvRTIEventRetractionHandle _rh：取消的代碼。

Return value：

None.

Exception：

- tvRTIInvalidRetractionHandle：企圖撤回不被 local federate 送出的事件，或者不在 local LRC 的過去的暫存區。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。
- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。

- tvRTIEventNotKnown：一個要求被產生去撤回一個 federate 不知道的事件。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

R.10 tvTimeBase:: TimeStep

Description：

以時間步進的方式前進到下一個時間點。

Synopsis：

```
tvRTIFedTime
tvTimeBase::
    TimeStep(
        bool available=false
    );
```

Input value：

- bool _available：設定是否要立即的前進到下一個時間。

Return value：

目前的模擬時間。

Exception：

- tvRTIInvalidFederationTime：一個對回呼或者服務的邏輯時間參數不是一個 federation 邏輯時間軸合法的點。
- tvRTIFederationTimeAlreadyPassed：企圖將存檔排程或者推進 federate 的邏輯時間到 federation 已經過的邏輯時間。
- tvRTITimeAdvanceAlreadyInProgress：在先前的時間進行服務仍未結束前引發另一個時間前進服務。
- tvRTIEnableTimeConstrainedPending：企圖在一個要求開啟的時間限制正在進行中，開啟時間限制或者推進時間。
- tvRTIEnableTimeRegulationPending：企圖在一個要求開啟的時間管理正在進行中，開啟時間管理或者推進時間。
- tvRTIFederateNotExecutionMember：一個在 federation 內容裡的合法引發服務在 RTI ambasador 與任何一個 federation 有關時被產生。
- tvRTIConcurrentAccessAttempted：當一個非再進入的服務的要求還在進行時被要求一個非在進入的服務。這個狀況發生在當一個 RTI ambassador 服務被從一個 federate ambassador 的回呼或是從同時地多執行緒所引發。

- tvRTISaveInProgress：當一個 federation 儲存正在進行中，企圖引發另一個儲存服務。
- tvRTIRestoreInProgress：當一個 federation 的復原還在進行時企圖引發一個服務。
- tvRTIRTIinternalError：一個 RTI 內部錯誤發生；詳細請查詢 federate 的紀錄檔案。
- tvRTITimeAdvanceAlreadyInProgress：在先前的時間進行服務仍未結束前引發另一個時間前進服務。
- tvRTITimeAdvanceWasNotInProgress：當沒有任何時間前進服務進行時，federate 被告知一個時間前進的允許。
- tvRTIFederateInternalError：一個 federate 的內部錯誤阻止他成功的進行一個 federate ambassador 回呼。

第五章、實例說明

為驗證所設計的類別程式庫的可行性，我們必須找一個符合 HLA 規格的模擬軟體來試驗，在我們可獲得的資源情況下，所得到符合 HLA 架構的模擬案例為 Sushi Restaurant 的範例，此範例能夠很完整的表現出一個具體而微的模擬環境以及執行結果。但是由於原本的範例程式是運用 Java 語言配合標準的 RTI 1.3 規則來製作的，因此我們以同樣的模擬場景與執行腳本將此範例改編為 C++版本，並配合我們所研究包裝出的程式庫，製作出一樣的模擬狀況。藉以驗證本計劃所設計的 RTI 程式庫的正確性以及方便性。

下面我們將分為兩個部份來解說：第一個部份為原本的 Sushi Restaurant 程式的執行方式；第二的部份為包裝後簡化了那些步驟，以及在使用上的程序。

第一部份、原版 Sushi Restaurant 執行方式解說

在開始解說原版的 Sushi Restaurant 程式之前，必須先說明一下在這個模擬環境中，程式預先定義了五個執行階段，分別是由五個 Synchronization Point 來控制是否進入這些階段，分別是：

1. Preliminaries：進入這個階段後，開始執行 time switch、publish、subscribe 等啟始的設定。

2. Populating：進入這個階段後，Object 會註冊一個 Instance，並且送出第一次的 Update。

3. Running：進入這個階段後，federate 會配合時間的推進要求開始執行 instance attribute 與 interaction 傳遞與接收。

4. Post-processing：進入這個階段代表目前已經接收到結束訊息，因此結束目前正在執行的動作，並等待所有成員結束工作。

5. Resigning：進入這個階段代表開始執行 unpublish、delete instance、resign federate 以及 destroy federation。

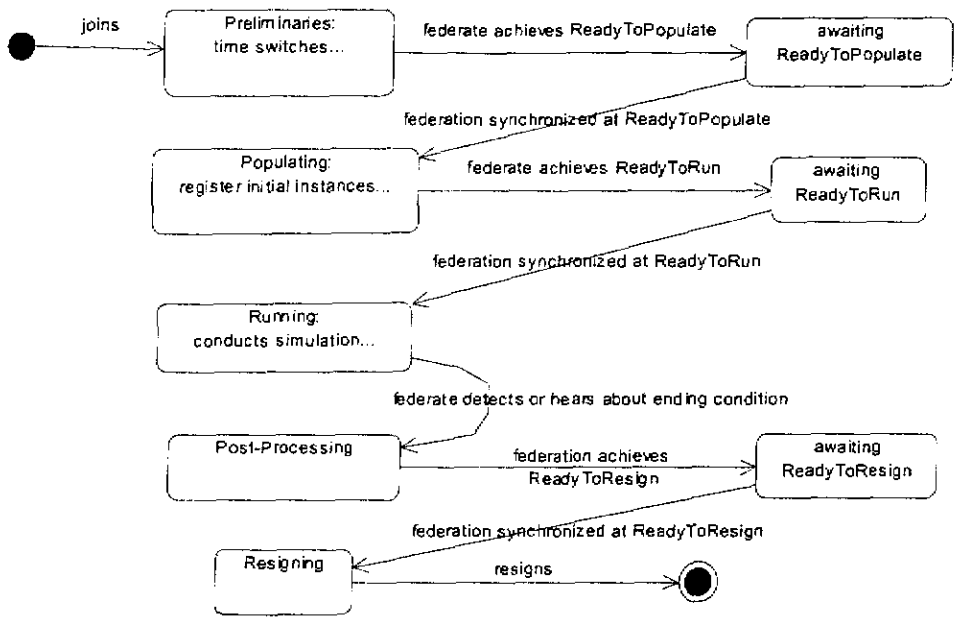


圖 5-1

由圖 5-1 來看，我們可以瞭解這五個階段的執行過程。接下來，我們將以這五個階段為基礎，分別詳述原版的 Sushi Restaurant 程式使用 RTI 1.3 版的程序。

1、Preliminaries：這個階段中的主要工作有下面的幾項，如圖 5-2 所示。

(1). 建立 Federation Execution：使用 createFederationExecution() 這個 function 來建立 Federation，如果要建立的 Federation 已存在則繼續到下一步驟。

(2). 加入 Federation Execution：使用 joinFederationExecution 這個 function 來使 federate 加入 federation。

(3). 註冊所有的 Synchronization Point：使用 registerFederationSynchronizationPoint 來分別針對五個 Synchronization Point 向 RTI 提出註冊，並藉由 tick 來驅動 RTI 的 callback。並接收 confirmSynchronizationPointRegistration 以及 announceSynchronizationPoint 兩個 Function 的 callback。在註冊確認及加入後，也同時正式進入 preliminaries 階段。

(4). 設定 Federate 的時間屬性：透過執行 enableTimeConstrained 或 enableTimeRegulation 來告知 federate 的時間屬性，接著以 tick 來驅動 timeConstrainedEnabled (或 timeRegulationEnabled) callback 告知時間屬性設定被接受。

(5). 以 getObjectClassHandle、getAttributeName、getInteractionClassHandle 以及 getParameterHandle 等 function 來取得 FED 當中制定的物件之 Name 及 handle 值。

(6). Federate 做出對 Object 的宣告：以 publishObjectClass 或 subscribeObjectClassAttibute 來做出宣告，並以 tick 趨動接收 startRegistrationForObjectClass 的 callback 來向 Object 註冊。

(7). Federate 做出對 Interaction 的宣告：以 publishInteractionClass 或 subscribeInteractionClass 對 RTI 做出宣告，並以 tick 趨動接收 turnInteractionOn 的 callback 來開啟 Interaction 的傳送開關。

(8). Preliminary 階段準備結束，並等待進入 populating 階段：以 synchronizationPointAchieved 告知 RTI 已經完成 preliminary 階段的工作，並送出 tick 等待 RTI callback federationSynchronized 通知可以往下一個階段前進。

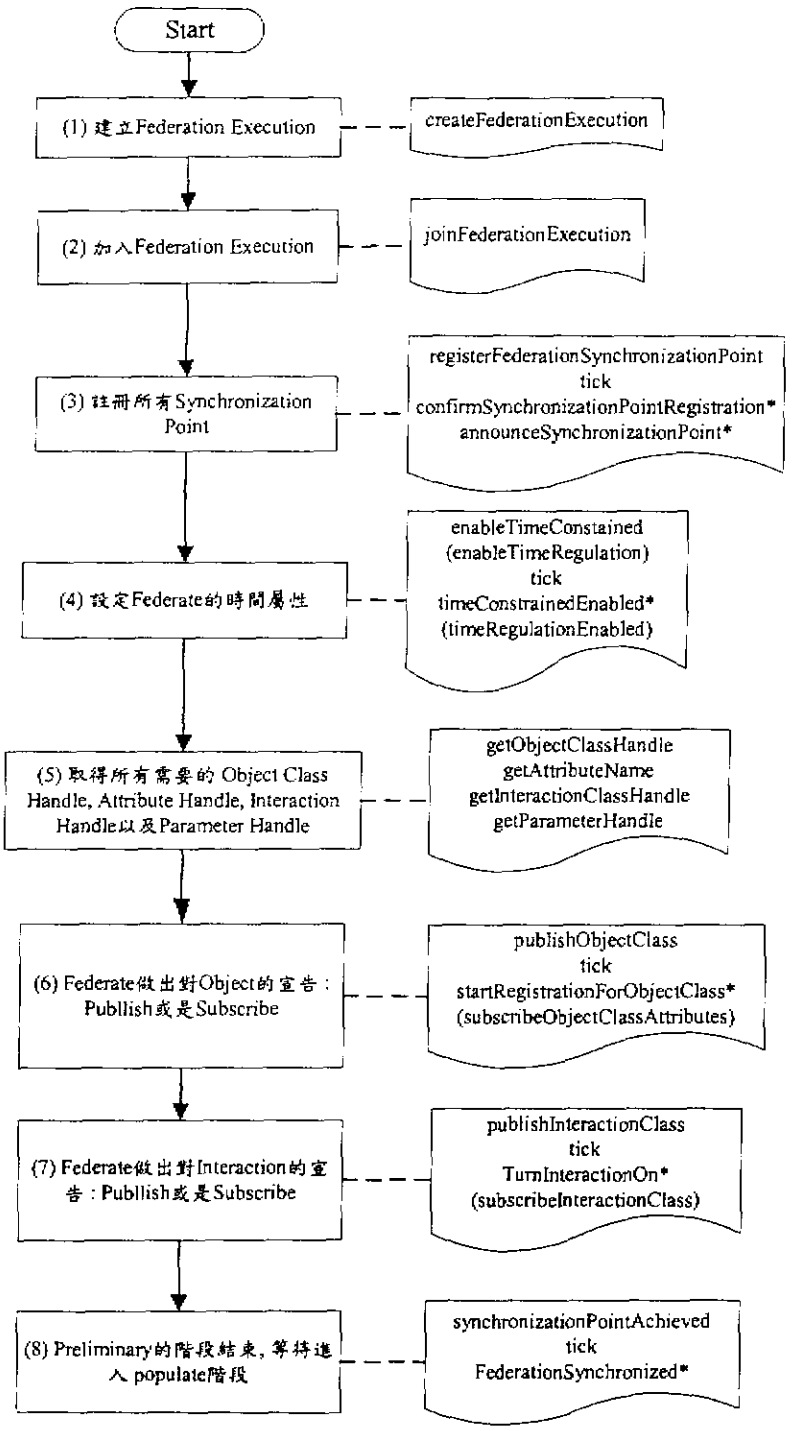


圖 5-2

在圖 5-2 內每個執行步驟的右邊都有一個 Document 符號，此符號內註明該步驟所需要用到的 HLA 規格的函式呼叫名稱，藉此可以清楚得知每個執行步驟的真正執行程序。以下的圖表將繼續沿用此習慣。

2、 Populating：這個階段中，Object 將會註冊其 instance，也就是實體化，所有執行步驟如圖 5-3 所示。

(1). 進入 Populate 階段。

(2). Register Object Instance 並做第一次的 update 動作：透過 registerObjectInstance 將 Object 實體化。並以 updateAttributeValues 這個 function 將初始的資料送出。

(3). Populate 階段準備結束，並等待進入 Running 階段：以 synchronizationPointAchieved 告知 RTI 已經完成 populate 階段的工作，並送出 tick 等待 RTI callback federationSynchronized 通知可以往下一個階段前進。

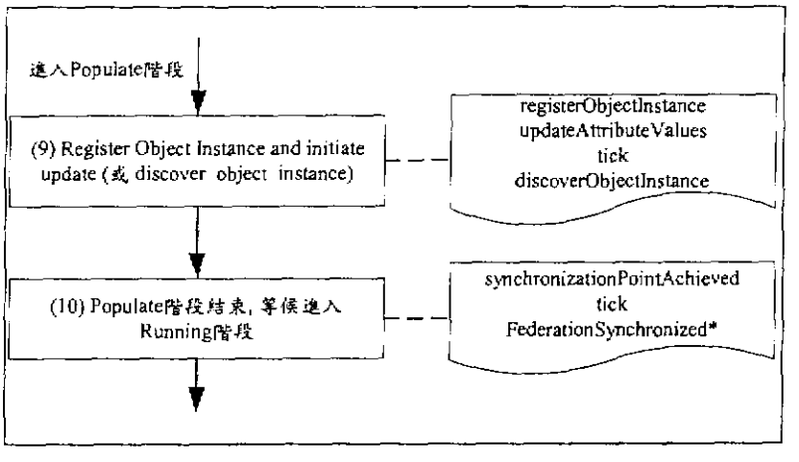


圖 5-3

3、Running：進入 Running 階段後，則開始執行 Running loop，在開始時先判斷是否有收到任何結束訊息或是已經執行至模擬結束的限制條件，如果有收到的話，則往決策的左邊"是"的方向前進離開 Loop，若決策結果是"否"的話，直接進入 Running loop 的下一步驟。而第二個決策是在 Running loop 中會遇到的幾種執行狀態，如下圖 5-4 所示。

(1). 第一種：以 updateAttributeValue 來處理內容資料的更新，接著要求時間的推進(透過 timeAdvanceRequest 或 nextEventRequest)，然後以 tick 趨動 update 值的傳出。

(2). 第二種：為 federate 要求 RTI 更新資料，透過 requestAttributeValueUpdate 這個 Function 來提出要求，同樣的，接下

來要求時間的推進(透過 timeAdvanceRequest 或 nextEventRequest)，然後以 tick 驅動 callback function reflectAttributeValue 的回傳值到達。

(3). 第三種：透過 sendInteraction 這個 function 送出 Interaction，接下來要求時間的推進(透過 timeAdvanceRequest 或 nextEventRequest)，並以 tick 驅動。

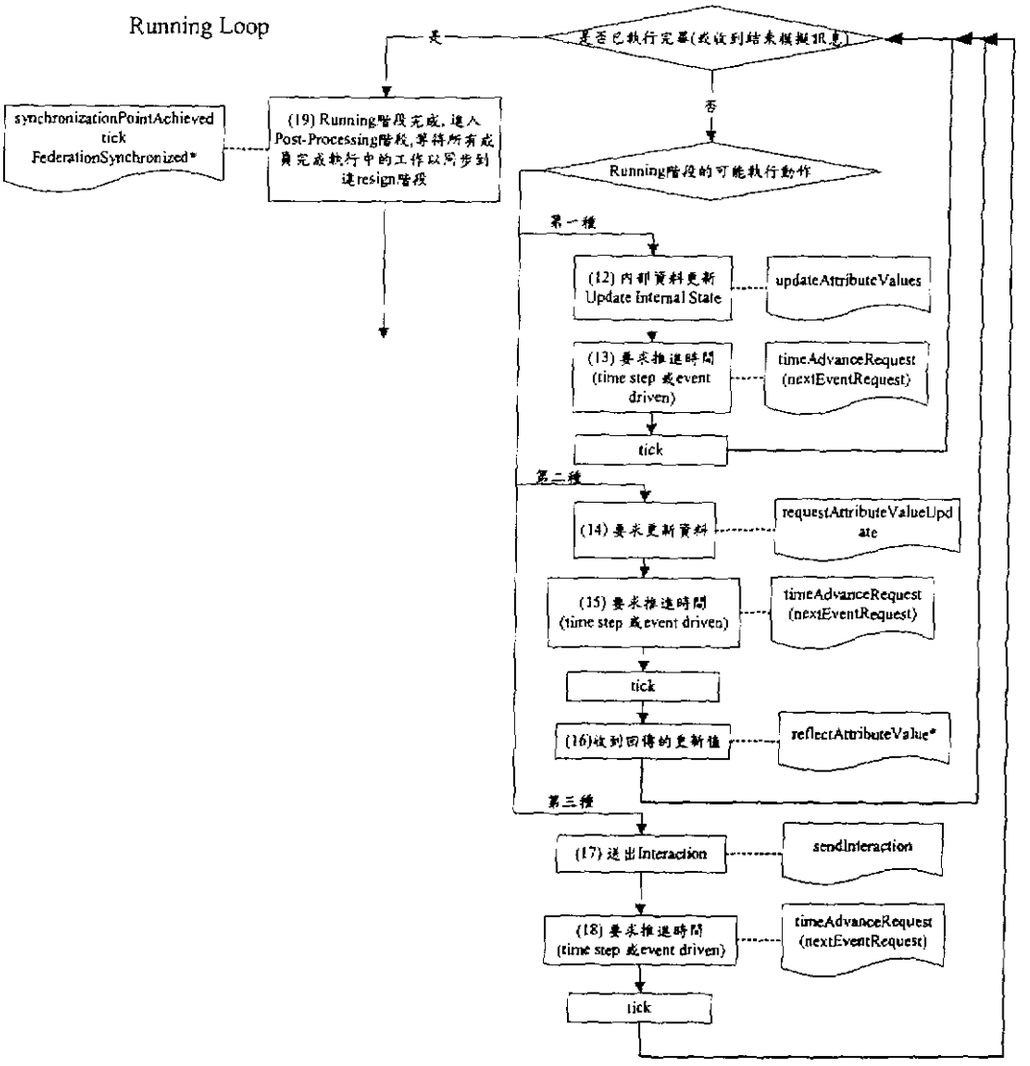


圖 5-4

4、Post-Processing：這個階段的執行，主要是結束所有目前的工作，並等待所有 federation 成員完成工作，以便進入下一階段，如圖 5-5 所示。

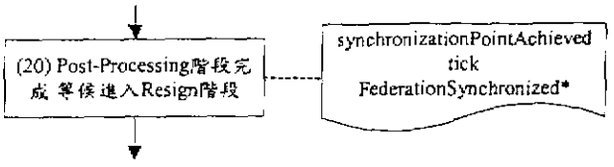


圖 5-5

5、Resign：這個階段，所有 federate 將開始移除物件並離開，如圖 5-6 所示。

(1). 解除 Interaction 宣告：透過 unPublishInteractionClass 這個 function。

(2). 解除物件宣告：透過 unPublishObjectClass 或 unSubscribObjectClass 的 Function。

(3). 刪除物件：以 deleteObject 這個 function 來刪除物件。

(4). federate 離開： 在物件都移除後，federate 透過 resignFederate 離開 federation。

(5). 結束 federation：最後離開的一個 federate 必須執行 destroyFederationExecution 來結束整個模擬。

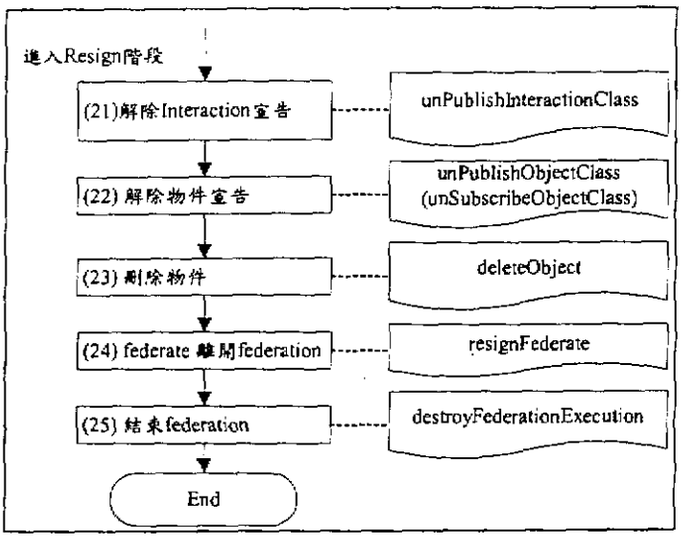


圖 5-6

圖 5-7 為將上述五大執行階段整合後的整體 Sushi 餐廳範例的流程圖。

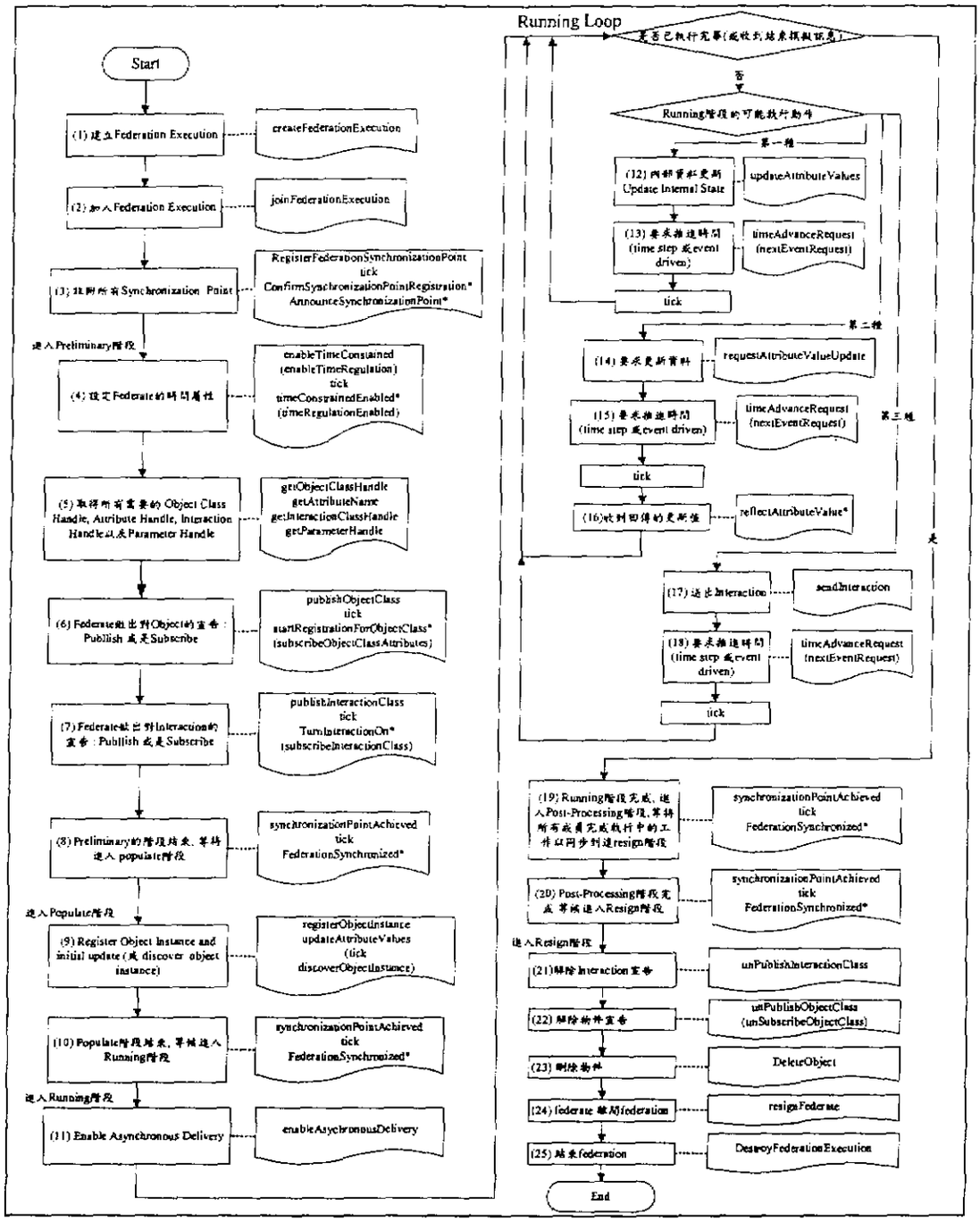


圖 5-7 Sushi Restaurant 運用 RTI 1.3 API 的執行過程全圖

第二部份、Sushi Restaurant 案例透過本案的 RTI 程式庫的執行

在下圖 5-8 中我們將圖 5-7 的流程圖以 3D 立體矩形方塊將分析包裝的類別函式框起來以了解所設計的程式庫是否滿足需求？同時可以與圖 5-7 做一個整體性的比較。

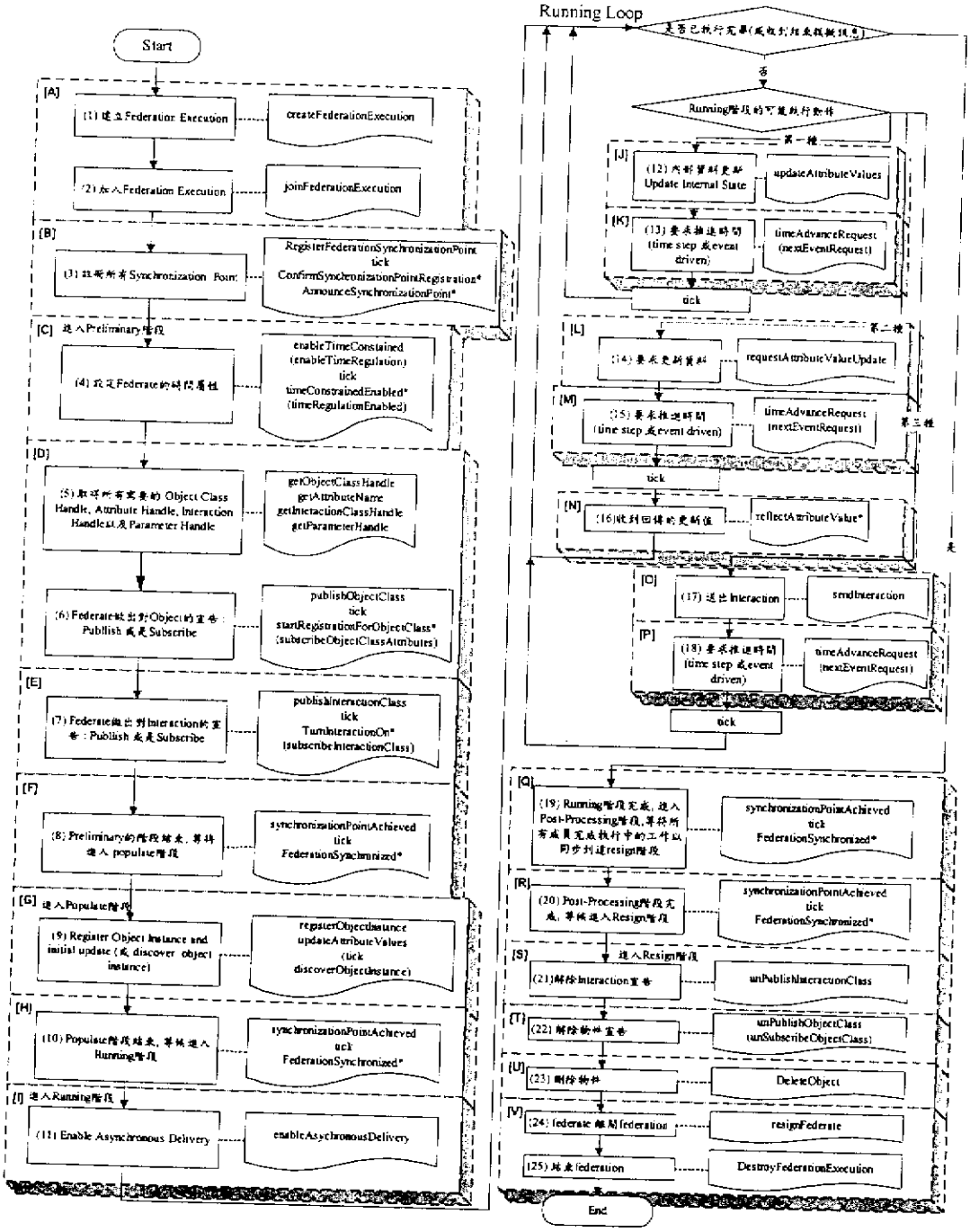


圖 5-8

圖 5-8 與圖 5-7 的差別是在於圖 5-7 所述為整體流程所會使用到的所有 HLA 介面函式，而圖 5-8 內每個 3D 立體矩形方塊代表所設計之類別程式庫的一個函式。各 3D 立體矩形方塊所代表的類別程式庫函式如下說明：

☑ 立體方塊 A - FederateBase 類別的衍生類別（tvFedeExec）的實體化(instance)過程建構子將會自動執行。

☑ 立體方塊 B－呼叫 tvFedeExec 的成員函式 Syn_Registeration()。

☑ 立體方塊 C－呼叫 tvFedeExec 繼承自 tvTimeBase 類別的解構子。

☑ 立體方塊 D－實體化(instance)tvEntity 類別時啟動 tvAttributeDeclare 類別的建構子。

☑ 立體方塊 E－實體化(instance)tvInteraction 類別時繼承自 tvInteractionDeclare 類別的建構子。

☑ 立體方塊 F－呼叫 tvFedeExec 的成員函式 Syn_Registeration()。

☑ 立體方塊 G－實體化(instance)之後由 tvEntity 類別的建構子完成。

☑ 立體方塊 H－呼叫 tvFedeExec 的成員函式 Syn_Registeration()。

☑ 立體方塊 I－呼叫 tvFedeExec 繼承自 tvTimeBase 類別的成員函式 Delivery()。

☑ 立體方塊 J－呼叫 tvEntity 類別的成員函式 Send()。

☑ 立體方塊 K－呼叫 tvFedeExec 繼承自 tvTimeBase 類別的成員函式 TimeStep()或 EventTime()。

☑ 立體方塊 L－呼叫 tvEntity 類別的成員函式 RequestValue()。

☑ 立體方塊 M－呼叫 tvFedeExec 繼承自 tvTimeBase 類別的成員函式 TimeStep()或 EventTime()。

☑ 立體方塊 N –tvFedAgent 呼叫 tvEntity 類別的虛擬成員函式 Receive()。

☑ 立體方塊 O –呼叫 tvInteraction 類別的成員函式 Send()。

☑ 立體方塊 P –呼叫 tvFedeExec 繼承自 tvTimeBase 類別的成員函式 TimeStep() 或 EventTime()。

☑ 立體方塊 Q –呼叫 tvFedeExec 的成員函式 Syn_Achieve()。

☑ 立體方塊 R –呼叫 tvFedeExec 的成員函式 Syn_Achieve()。

☑ 立體方塊 S –刪除 tvInteraction 的實體時繼承自 tvInteractionDeclare 的解構子自動執行。

☑ 立體方塊 T –刪除 tvAttributeDeclare 實體時自動啟動解構子。

☑ 立體方塊 U –刪除 tvEntity 實體時自動啟動解構子。

☑ 立體方塊 V –程式結束時自動執行 tvFederationBase 的解構子。